# Pedagogical Content Knowledge and Educational Cases in Computer Science: An Exploration

***H. Koppelman***
***University of Twente, Enschede, the Netherlands***
***Open University of the Netherlands, Heerlen, the Netherlands***

**H.Koppelman@utwente.nl**

## Abstract

The concept of pedagogical content knowledge has been explored in the context of several disciplines, such as mathematics, medicine and chemistry. In this paper the concept is explored and applied to the subject matter of computer science, in particular to the subdomain of building UML class diagrams. It is argued that the identification and analysis of problems that students experience with important concepts should be at the heart of pedagogical content knowledge in this subdomain, as well as the description of pedagogically rich exercises for tackling those problems. For two examples, relevant pedagogical content knowledge is identified and represented in the form of educational cases.

**Keywords**: Pedagogy, computer science education, UML class diagrams, exercises.

## Introduction

Different categories of knowledge are needed for teaching. Of course, instructors need subject matter knowledge in the first place, but they also need pedagogical knowledge, such as knowledge of teaching techniques, and knowledge. In this paper another, specialized kind of knowledge is studied: pedagogical content knowledge, a term that was introduced by L. Shulman (1986). The concept of pedagogical content knowledge may be introduced by the words of Shulman himself. This category of knowledge includes:

> ' … the most useful forms of representation of … ideas, the most powerful analogies, illustrations, examples, explanations, and demonstrations – In a word, the ways of representing and formulating the subject that make it comprehensible to others. … Pedagogical knowledge also includes an understanding of what makes the learning of specific topics easy or difficult: the conceptions and preconceptions that students … bring with them …. If those preconceptions are misconceptions, which they so often are, teachers need knowledge of the strategies most likely to be fruitful in reorganizing the understanding of learners, because those learners are unlikely to appear before them as blank slates.' (Shulman, 1986)

Pedagogical Content Knowledge can be seen as the intersection between pedagogy and content. Therefore, pedagogical content knowledge is a practical way of knowing the subject matter, which is

used by instructors when they are teaching. It is a form of instructors' professional knowledge that builds upon but is different form teachers' subject matter knowledge. Pedagogical content knowledge can be considered as subject matter knowledge *for teaching*.

This knowledge develops gradually, mostly through practice, and builds up in the course of many years of face-to-face interaction with students. Experienced instructors possess it, but novice instructors usually do not.

It makes the difference between the way experienced instructors know their subject matter and the way novice instructors do.

The identification of pedagogical content knowledge is not only a matter of theory and educational research, but also has practical consequences. One of these is that this knowledge can be used to prepare new generations of instructors. In many cases, especially in the context of higher education, novice instructors often find themselves teaching by accident, without much pedagogical knowledge and without much support. (Eckstein, Bergin, & Sharp, 2001) Typically, a person with the specific subject matter knowledge that is required will be asked to teach it. But having subject matter knowledge is very different from having pedagogical content knowledge. Consequently, it would be very useful if there were efficient ways to facilitate the sharing of this knowledge between experts and novice instructors.

In order to use pedagogical knowledge for sharing, it is not enough just to identify it. These collected experiences of the teaching community should also be made available in some useful representation. Therefore an efficient and effective vehicle for sharing this knowledge is needed. Shulman (1986) argues that a common way to represent pedagogical knowledge is in the form of propositions. An example of a proposition is: "Break a large piece of chalk before you use it for the first time, to prevent squeaking against the board." Propositions are an efficient way of representing knowledge, but one drawback of propositions is that they are hard to remember and that they are decontextualized and devoid of detail. For these reasons Shulman introduces the idea of educational cases. These can be used to organize pedagogical content knowledge in a coherent way in a body of practical knowledge: 'Cases may be examples of specific instances of practice – detailed description of how an instructional event occurred – complete with particulars of contexts, thoughts and feelings. On the other hand, cases may be exemplars of principles exemplifying in their detail a more abstract proposition or theoretical claim.' (Shulman, 1986)

Educational cases can contain several of the mentioned aspects of pedagogical content knowledge in a coherent way: student problems and misconceptions, an analysis of those problems and misconceptions, and an instructional strategy to overcome those problems.

Therefore these cases document what can be done in particular situations and why. They can contain details and a context.

Educational cases are generated through experience and reflection. They can document experienced instructors' 'wisdom of practice'. For that reason these cases can be used to prepare new generations of professionals.

An example of a possible case containing pedagogical content knowledge is given by L. S. Shulman (1992). The domain is teaching arithmetic to children. The (fictional) case could start with describing the problems that children have to understand what it means to divide by zero (which is not possible, of course). It gives an explanation of the 'subject matter' of dividing by zero and gives an account of the common mistakes children make and the recurrent difficulties they have. It might show examples of the errors children make. The case presents an instructional strategy to remedy these problems and shows the results of that strategy.

The concept of pedagogical content knowledge has been explored in the context of several disciplines, for example in mathematics, biology and chemistry (Bucat, 2004; Ma, 1999; Shulman, 2004). Each discipline has its particular pedagogical content knowledge, because this knowledge is at the intersection of content and pedagogy. This paper explores the concepts in the context of computer science. The main question is: how can the concept of pedagogical content knowledge be characterized and applied in the discipline of computer science? To study this question, in the next section characteristics of student activities in computer science are analysed, as well as characteristics of expert instructors' behavior. The findings are applied to the subdomain of UML diagram techniques, as used in object-oriented analysis and design. Two educational cases, containing pedagogical content knowledge in this field, are described.

# Pedagogical Content Knowledge in Computer Science

How can the pedagogical content knowledge in the domain of computer science be characterized? An essential aspect of the work of computer scientists is *solving problems*. The results of this process of problem solving are all kinds of artifacts: diagrams, programs, specifications, proofs, and so on.

An important observation is that students do not learn problem solving by reading about it or listening to somebody talking about it. Many reports within the computer science community support this observation. For example, Eckstein et al. observe: 'Passive students don't learn much. If students listen to explanations, without becoming engaged, what is learned is unlikely to go into long-term memory. The consequences of a theory are unlikely to be obvious to one who reads about, or hears about the theory. The unexpected difficulties inherent in using the theory or applying the ideas are not likely to be apparent until you actually do use the theory.' (Eckstein et al., 2002).

The idea that students learn while being active is in agreement with the well-known approach of active learning. Active learning gets students involved in activity in the classroom rather than passively listening to a lecture. Examples of activities are discussing, solving problems and answering questions. Active learning has been applied to many disciplines, among them computer science. As has been observed (McConnell, 2005), computer science is by its very nature an active learning discipline. Already from the start of the discipline, active learning has been applied, especially in programming projects. More recently several active learning techniques have also been used inclass. As a consequence several reports exist about experiences with active learning techniques and strategies in computer science education (Briggss, 2005; McConnell, 1996; Whittington, 2004).

Therefore it should be expected that students learn most in a problem solving context when instructors give students problems in the form of exercises and cases to train their problem solving activities. And indeed, performing such activities is what students actually do in many topics in the discipline of computer science, like programming, database design and systems analysis and design.

The role of the instructors in this context is, in the ideal case, knowing the hard modelling questions, the problems the students experience, the common mistakes, the usual misconceptions. Instructors carefully compose or select relevant exercises that clarify and tackle these problems and from which the students can learn as much as possible. They know these exercises thoroughly and foresee which correct and incorrect solutions the students might conceive. If indeed students come up with incorrect or partially correct solutions, the ideal instructors know which additional questions to ask and which answers to provide. They can also control complexity by providing examples and exercises that will give students a valid and complete picture and yet are as simple as possible.

From this overview of the pedagogical content knowledge of experienced computer science instructors, it can be concluded that a lot of their knowledge can be linked to exercises. For this reason exercises are at the heart of the educational cases that are discussed in the next section. The exercises give a framework that can be used to bind chunks of pedagogical knowledge together in a coherent body of knowledge.

From a pedagogical point of view, students' misconceptions of students are very interesting. Shulman observes: 'The study of student misconceptions and their influence on subsequent learning has been among the most fertile topics for cognitive research. We are gathering an ever-growing body of knowledge about the misconceptions of students and about the instructional conditions necessary to overcome and transform those initial conceptions. Such research-based knowledge, an important component of the pedagogical understanding of subject matter, should be included at the heart of our definition of needed pedagogical knowledge.' (Shulman, 1986)

The approach of learning-from-mistakes is in the context computer science education advocated by Ginat (2003). He proposes to 'plan a lesson based on expected student errors, in order to "make a point", illustrate a principle, or affect student beliefs."

As a consequence, we decided to give misconceptions and mistakes, and analyses of them, an eminent role in the pedagogical content knowledge.

In the next section, two examples of educational cases capturing pedagogical content knowledge are described.

# Examples of Educational Cases

This paper explores pedagogical knowledge that is connected to the subject matter of using UML techniques. UML is well-known to be hard to learn for novices, because it has many concepts, with complex semantics (Hansen & Ratzer, 2002). Raner (2000) states, for that reason, that UML is excellent for experts, but bad for novices.
One of the diagram techniques of UML is drawing class diagrams (Frosch-Wilke, 2003). Using class diagrams has become central within object-oriented analysis and design. The class diagram is not only widely used, but it also has the greatest range of modelling concepts. Therefore, it is not surprising that Frosch-Wilke found in an empirical study (2003) that many students have problems identifying appropriate classes. A major reason is the lack of a 'recipe' for doing this.
During several years of lecturing we noticed many specific problems students experience building class diagrams. Frequently occurring problems concern:

- understanding the difference between the concepts class and instance
- identifying appropriate classes in given requirements
- identifying appropriate relations between given classes
- understanding cardinalities of relations
- fully understanding the concept of generalization
- understanding and applying the concept of association class
- modelling different relations between the same classes
- redundant associations.

Many problems on this list were also found in an empirical study (Frosch-Wilke, 2003), about problems with using class diagrams (among other things).

Two problems are analysed in this section. The analysis is based upon our experiences of teaching object-oriented analysis and design over many years.

## *Case 1: Using an Association Class*

For many novices the key concept of an association class in UML is a hard one. For them it is hard to see *why* in certain cases an association class should be used. They are inclined to overlook the use of it, or to use an independent class in an incorrect way. But even if they are told *that* an association class is appropriate, they cannot figure out *how* to use it. Therefore, as an introduction it is appropriate to deal with these problems one by one, by first focusing on the why question. This can be done by giving them an exercise that can elicit a discussion about the 'why' of association classes. Therefore, give the students a problem statement, give them a class diagram with an association class, and give relevant attributes; ask the students to allocate the attributes to the given classes.

### Example

A guest stays in a hotel room. Figure 1 shows a UML class diagram of this situation. Relevant classes are Guest, HotelRoom and Stay. In this example Stay is the association class.

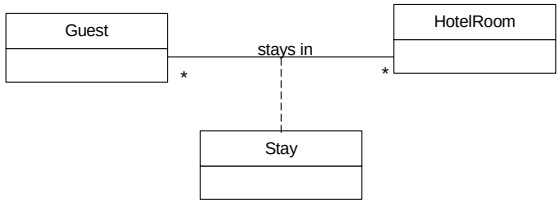


**Figure 1: Class diagram with an association class**

This diagram is offered to the students, as well as a number of attributes: *room number*, *rate*, *way of paying*, *guest number* and *date of arrival*. The assignment for the students is to add these attributes to the classes in the class diagram in Figure 1.

For most attributes in the example it is quite clear to which class they should be allocated. But the exercise is constructed in such a way that there can be relevant discussions about the allocation of some attributes, for example *rate*. There is not enough information to make a definite choice (For example, is *rate* the standard rate for the room, or is it the rate that the guest is going to pay?), but the consequences of the alternatives can be examined. The same goes for *way of paying*. It can be discussed what the consequences are of allocating it to *Guest* or to *Stay*.

Those discussions clarify why the use of an association class is adequate in this example.

Diagrams with an association class can be transformed into an equivalent diagram without an association class. A relevant question is to give this equivalent class diagram and ask the students whether semantic differences exist between both diagrams.

In the example this question applies to the diagram in Figure 2. As an alternative this topic can be posed in an open question: is there an equivalent diagram without the use of an association class?

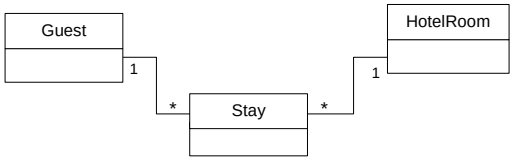


**Figure 2: Class diagram without association class.**

The goal of this problem is to generate discussions about *why* the use of association classes can be appropriate. *How* to use association classes, has been given in this problem. For many novices it is not only hard to understand the 'why' of association classes, but also the 'how'. The same type of exercises can be used to focus on the 'how', but, of course, without offering the students the class diagram.

## *Case 2: Using a Description Class*

Often you have a set of objects that share common information but also differ in important ways. One example is the set of copies of the same book in a library. These copies have the same author and ISBN, but different bar-code identifiers and they are borrowed by different people. In this case it is useful to introduce a description class (which is also called an abstraction class or a specification class) that contains all the data that are common to the set of objects. Novices have problems in seeing when it is appropriate to use description classes. One of the problems is that they confuse the concepts of super class and description class.

Therefore we give the students an exercise in which using a description class is appropriate and which can elicit discussions about the concept of description class. An example will be given.

### Example

A chain of hotels consists of several hotels in different cities. Each hotel has rooms, with distinct numbers. All hotels have 3 types of rooms: business class, standard and economy. Every type has a fixed standard rate. (These properties are the bare minimum. Others can be added, of course)

The assignment for students is to model this situation by way of a class diagram. Figure 3 shows the correct class diagram.

When modelling this situation many students do not come up with a description class. In the example they allocate all properties of the rooms to a class *HotelRoom*, missing the desirability of a class *RoomType*. In this case the instructor discusses the usual problems of ignoring the description class: risk of inconsistency, inefficiency, and possible loss of information.

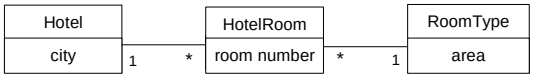


**Figure 3: Class diagram with a description class**

Many students will confuse the use of a description class and generalisation. In the example they will come up with subclasses as *BusinessClass*, *Standard* and *Economy*. The instructor explains that this is not a good idea, because the distinguished subclasses do not have different properties.

Crucial is that many instances have exactly the same information. This is an indication that a description class should be used.

The concepts generalisation and using a description class are easily confused. Therefore it is advisable that the instructor provokes a discussion about this confusion, if the students themselves do not do this. To show the difference clearly the instructor can extend the exercise with generalization. In the example a possible extension is: the hotel not only rents hotel rooms but also congress rooms. Congress rooms have a number, like hotel rooms, but they also have a seating capacity, unlike hotel rooms. Therefore it is appropriate to introduce a super class *Room* with subclasses *HotelRoom* and *CongressRoom*. Such an example clearly shows the difference in a very simple way.

In many cases using a description class is good modelling practice, but not strictly necessary. In such cases it takes some effort to convince the students why this is desirable and maybe it depends on the persuasiveness of the instructor whether he or she succeeds. In these cases the instructor can extend or change the exercise, in a way that makes the use of a description class inevitable. In the example the exercise can be extended with the reservation of rooms. If a guest makes a reservation for a room, the system initially only makes a reservation for the type, not for an actual room (Figure 4). There is no association between reservation and *HotelRoom*. The actual room is assigned only shortly before the arrival of the guest.

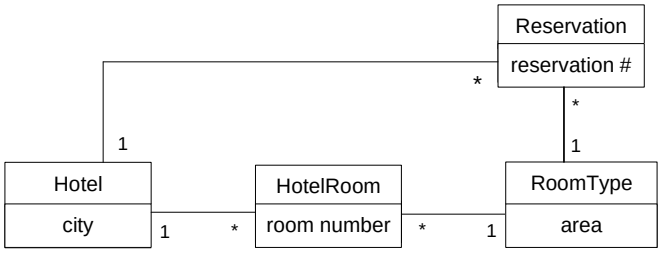


**Figure 4: Class diagram that cannot do without a class RoomType**

Recently many analysis patterns have been found and published, for example by Fowler (1997). The construction described in this educational case is a well-known analysis pattern, called *Item-Description* pattern or *Abstraction-Occurrence* pattern. Therefore, this case suggests that the instructor should discuss the *Item-Description* analysis pattern.

# Discussion

Pedagogical content knowledge is based on the wisdom of experience. It is not invented but it is found. It describes common practice. Therefore first, provisional formulations of it should be reviewed in a public setting by a group of peers of the authors. In this way ideas can be tightened up, more knowledge can be added and formulations can be improved.

In this paper pedagogical content knowledge has been represented in the form of educational cases. In the computer science society another tool for documenting knowledge has become well known: patterns. Patterns have been used to represent design knowledge in the first place, but also to represent pedagogical knowledge. The result is a collection of useful pedagogical patterns. Examples are patterns about giving feedback, supporting active learning, supporting experiential learning, initial course design, and so on (Bergin, 2001; Eckstein et al., 2002; Fincher, 1999). These patterns express general pedagogical knowledge. Much less work has been done on expressing pedagogical content knowledge in patterns. In this paper educational cases have been

used to document pedagogical knowledge, but pedagogical patterns could also have been used. We chose cases in order to join activities in other disciplines such as mathematics where cases have been used to document this kind of knowledge. But it is interesting to explore the suitability of other representations.

Thus documented knowledge can facilitate the acquisition of pedagogical skills of those who need them. Novice instructors could profit from capturing and sharing pedagogical knowledge, in the first place. Thus teacher education could benefit from this knowledge. But experienced instructors could also learn from each other. Every instructor has his or her own private 'killing' exercises and examples. If these could be exchanged, peers could learn from each other.

But this is not the end of the story. The explicit availability of pedagogical knowledge is not only useful for those who actually teach in the classroom. It would be of great usefulness if the experiences of skilled instructors in face-to-face education could be made explicit, and be at the disposal of developers and instructors of distance education.

Distance education usually has hardly any face-to-face interaction or online communication. Students should be able to complete these courses on their own without much help or instruction. This implies that the courses should be carefully designed. It is a challenge to devise distance education courses that incorporate a lot of pedagogical knowledge in an efficient way in printed materials and within the context of interaction in an electronic environment.

# Conclusions

In this paper the concept of pedagogical content knowledge has been applied to the subdomain of designing UML class diagrams. Within this subdomain the concepts of association class and description class are analysed. The knowledge that has been documented in this paper consists in the first place of:

- the identification and analysis of problems that students experience with two concepts
- the description of pedagogically rich exercises, for tackling those problems
- suggestions for the feedback to give to students struggling with those concepts and for eliciting meaningful discussions about those concepts.

The pedagogical knowledge as described here originates from the experience of skilled instructors. Some of this experience can also be found in publications such as textbooks and practitioners reports. Published results of pedagogical computer science educational research (for example (Bergin, 2001)) are another possible source for finding relevant pedagogical knowledge.

The approach seems suitable for subdomains where problem solving is a core activity, such as analysis and design, programming, relational databases and SQL, mathematics for computer science, logic and so on.

# References

Bergin, J. (2001). A pattern language for initial course design. *ACM SIGCSE Bulletin, 33*(1), 282-286.

Briggs, T. (2005) Techniques for active learning in CS courses. *Journal of Computing Sciences in Colleges, 21*(2), 156-165.

Bucat, R. (2004). Pedagogical content knowledge as a way forward: Applied research in chemistry education. *Chemistry Education: Research and Practice, 5*, 215-228.

Eckstein, J., Bergin, J., & Sharp, H. C. (Eds.). (2002). Patterns for active learning. *PLoP 2002*. Retrieved March 11, 2008, from http://jerry.cs.uiuc.edu/~plop/plop2002

Fincher, S. (1999). Analysis of design: An exploration of patterns and pattern languages for pedagogy. *Journal of Computers in Mathematics and Science Teaching*, *18*(3), 331-348.

Fowler, M. (1997). *Analysis patterns: Reusable object models*. Reading, MA: Addison-Wesley Longman.

Frosch-Wilke, D. (2003). Using UML in software requirements analysis – Experiences from practical student work project. *Proceedings of the 2003 Informing Science + IT Education Conference*, Pori (Finland), 175-183. Retrieved March 11, 2008, from http://www.informingscience.org/proceedings/IS2003Proceedings/docs/032Frosc.pdf

Ginat, D. (2003). The greedy trap and learning from mistakes. *SIGCSE Bulletin*, *35*(1), 11-15.

Hansen, K. M., & Ratzer, A. V. (2002). Tool support for collaborative teaching and learning of object-oriented modeling. *ACM SIGCSE Bulletin*, *34*(3), 146-150.

Ma, L. (1999). *Knowing and teaching elementary mathematics*. New Jersey: Lawrence Erlbaum Associates.

McConnell, J. J. (1996). Active learning and its use in computer science. *ACM SIGCSE Bulletin, 28*(SI, June), 52-54.

McConnell, J. J. (2005). Active and cooperative learning: tips and tricks (part I). *ACM SIGCSE Bulletin, 37*(2), 34-38.

Raner, M. (2000). Teaching object-orientation with the Object Visualization and Annotation Language (OVAL). *ACM SIGCSE Bulletin*, *32*(3), 45-48.

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher, 15*(2), 4-14.

Shulman, L. S. (1992). Merging content knowledge and pedagogy: An interview with Lee Shulman by Dennis Sparks. *Journal of Staff Development, 13*(1).

Shulman, L. S. (2004). *The wisdom of practice: Essays on teaching, learning, and learning to teach*. San Francisco, CA: Jossey-Bass.

Whittington, K. J. (2004). Infusing active learning into introductory programming courses. *Journal of Computing Sciences in Colleges*, *19*(5), 249-259.

## Biography

**Herman Koppelman** is instructor in the department of Computer Science of the University of Twente and assistant professor in the department of Computer Science of the Open University of the Netherlands.