in Proceedings WCCE 2005, Cape Town, South Africa, Jul 4-7, 2005.

TEACHING OBJECT ORIENTED MODELLING WITH CRC-CARDS AND ROLEPLAYING GAMES

Börstler, J.[1], Schulte, C.[2]

[1]Umeå University, Department of Computing Science, Umeå, Sweden. E-mail: jubo@cs.umu.se
[2]Free University Berlin, Department of Computing Science, Berlin, Germany. E-mail: schulte@inf.fu-berlin.de

ABSTRACT

CRC-cards have been adopted by many educators to teach early design in their object-oriented programming courses. In this paper we discuss our experiences using the CRC-card approach in upper secondary school and undergraduate programming courses. Although our experiences are generally positive, we have noticed many problems and issues that have largely gone unnoticed in the literature. It is therefore important to apply the CRC-card approach very carefully to avoid building up or reinforcing misconceptions about object-orientation. In this paper we discuss two issues in details. One issue belongs to unintentionally mixing of learning goals. The other issue is the usage of CRC-cards as object surrogates during the scenario roleplay, which is a major source for class/object confusion. To avoid these problems, we introduced a new type of diagram to support the roleplay activities. We also propose to carefully distinguish two steps: Introducing the approach and the notations, and actually using CRC-cards as a modelling tool. The first step aims to introduce the CRC-cards approach itself and the use of CRC-cards to achieve a basic understanding of object oriented concepts. The cards are used as a learning medium to foster object-oriented thinking. In the second step, CRC-cards are used to gain understanding of a domain, learn how to analyse and conceptualise a domain, test a model with roleplay and to introduce more specific object-oriented concepts.

1. INTRODUCTION

CRC-cards have originally been described as a tool for teaching object-oriented thinking to experienced programmers (Beck & Cunningham, 1989). They are a simple informal tool for collaborative object-oriented modelling. Many educators have adopted CRC-cards as an approach to teach design early and convey a bigger picture of (systematic) software development (Biddle, Noble & Tempero, 2002; Börstler, Johansson & Nordström, 2002; Gray, Guzdial & Rugaber, 2002; Schulte, Magenheim & Niere et al., 2003). However, the approach has also successfully been applied outside the educational context (Bellin & Simone, 1997; Hvam, Riis & Hansen, 2003; Wirfs-Brock; Wilkerson & Wiener, 1990).

The power of the CRC-card approach lies in its associated roleplay activities. Scenario roleplay is usually used in the social (behavioural) sciences to evaluate human behaviour in specific yet hypothetical situations. The roleplay participants are assigned roles they enact according to predefined scenarios, much like actors following a script when playing the characters in play. During the roleplay, the participants learn a lot about themselves, the other participants, and the roles played. The power of the roleplay lies in its interactivity that supports creativity and sharing of knowledge. Roleplaying is an effective way to simulate or explore hypothetical situations, since the characters and scripts (scenarios) can be easily varied. In CRC-card roleplays, the characters are the objects in our system and the scenarios are situations of system usage. Scenario roleplay can be used in very early stages of software development, before any code is written (or even designed). That makes it appealing for novices as a means to familiarise with the object-oriented way of thinking, without needing to bother with syntactic details of actual programming or design languages.

The remainder of this paper is organised as follows. In the next section, we briefly introduce CRC-cards and Roleplays. In section 3, we evaluate the CRC-cards approach, briefly summarise our experiences, and argue for the use of Role-Playing Diagrams and the proposed two-steps approach, which are described in more detail in section 4. Section 5 concludes the paper.

2. A BRIEF INTRODUCTION TO CRC-CARDS AND ROLEPLAYS

2.1. CRC-Cards

CRC stands for Class, Responsibilities and Collaborators (Beck, 1993). A CRC-card is a standard index card that has been divided into regions, as shown in figure 1.

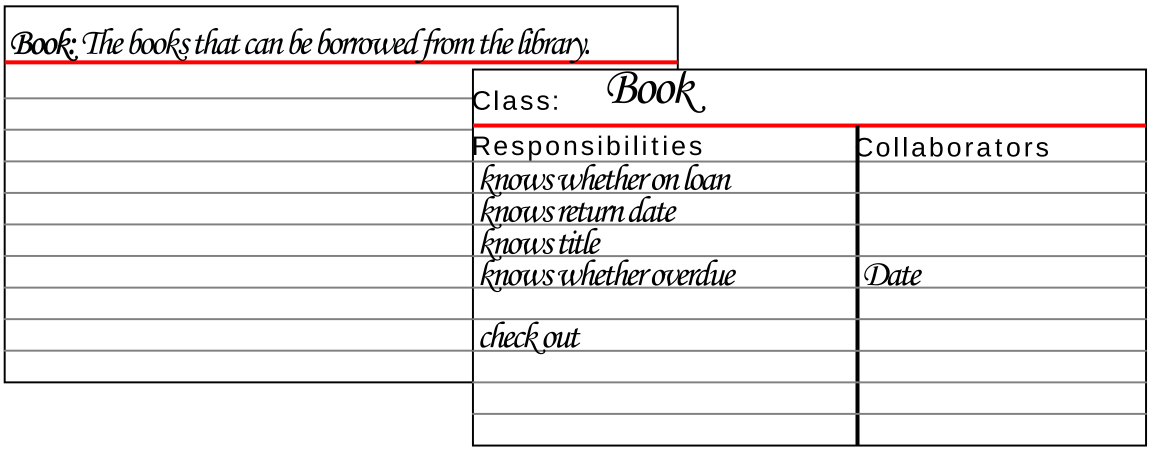

Figure 1: Example CRC-card for a class `Book`.

A CRC-card corresponds to a class. It describes the properties that certain kinds of objects of interest in the problem/ application domain have in common. An object can be a tangible thing, person, place, event, or (abstract) concept. A class should have a single and well-defined purpose that can be described briefly and clearly. It should be named by a noun, noun phrase, or adjective that adequately describes the abstraction. The class name is written across the top of the CRC-card. A short description of the purpose of the class is written on the back of the card.

A responsibility is something the objects of a class take care of; a service provided for other objects. A responsibility can be either to know something or to do something. A book object in a library for example might among other things be responsible for checking itself out and to know its title and whether it is out on loan. To do something an object usually uses its (local) knowledge. If this knowledge is not sufficient for the purpose, the class can require help from other objects (its collaborators, see below). The responsibilities of a class are written along the left side of the card.

The collaborators describe which kinds of objects can be asked for help to fulfil a specific responsibility. An object of a collaborator class can for example provide further information, or actually take over parts of the original responsibility (by means of the collaborators own responsibilities). A book object for example can only know whether it is overdue, if it also knows the current date. In figure 1, this information is provided by the collaborator `Date`. Collaborators are written along the right side of the card in the same row as the corresponding responsibility.

The back of the card can be used for the class' description, comments and miscellaneous details.

2.2. Roleplaying with CRC-Cards

The goal of the CRC-card approach is to develop, discuss and evaluate object-oriented models. In the roleplay activities, objects are treated as living entities that fulfil certain responsibilities in the context of the system to be developed. Students enact these objects in order to explore scenarios of system usage. They play the objects. When a request is sent to an object, the player tries to fulfil the request by acting out the corresponding responsibility, which is listed on the CRC-card. This can mean, that he calls another object (the collaborator on the card) for help. Thereby the roleplay participants follow responsibilities, sub-responsibilities, etc. and so make object-object collaborations alive. The roleplay gives learners a 'live impression' of the functionality of an object-oriented program. Because students are involved in playing this 'game,' commenting or discussing the steps they are actively engaged in, several sensorial inputs are used which helps them to remember things.

This makes CRC-cards particularly well-suited for collaborative learning. A group size of 4-6 people seems to work best. Smaller groups usually lack the right blend of different backgrounds. In larger groups, it becomes too difficult to reach a consensus. It is important not to get bogged down in discussions about implementation details. The goal is to discuss and evaluate different object-oriented models.

Scenario roleplay is a 'what happens next' game, in which scenarios are tested. Scenarios are usually started with a user request, so the teacher or a student acts as a user and starts the play by giving some 'user input' (see Biddle, Noble & Tempero, 2002).

While acting out scenarios, students have to follow the rules of an object-oriented program. For example, each object operates solely based on the responsibilities described on its corresponding CRC-card. In collaborations, only objects can be called, which are known to the caller. So maybe before acting out a scenario, the involved objects have to be created and linked together. Relevant object states and their changes must be remembered in order to act out the responsibility according to the given scenario.

In order to make it easier to follow these steps one might use some kind of token to indicate sending of request (like throwing a ball) and to mark the actual active object (for example by holding up the corresponding card). While this helps, it fails to provide a history cache allowing to step back to the caller or to recover from dead ends. We propose the use of RPDs as a tool to keep track of the single steps.

In order to play a roleplay properly, students have to keep much detailed information in their minds (and the teacher, of course, also). So we suggest using additional diagrams to keep track of important information revealed during a role-play, and to draw conclusions from the result.

It's important that the teacher carefully outbalances between smoothness and failure correction during roleplay (a common problem in learning a foreign language). Another challenge is to prevent students from getting lost in discussing details on the one hand, and from overlooking important aspects on the other hand.

3. PEDAGOGICAL EVALUATION OF CRC-CARDS

The literature about using CRC-cards in the teaching of object-orientation to novices is generally positive. Problems are reported rarely, and if so they are not considered problematic (Biddle et al., 2002). To take the best pedagogical advantage from the approach we discuss some issues using CRC-Cards reported in literature and from our own experiences in order to constitute our proposal for a systematic approach to CRC-Cards.

A major problem of the CRC approach is that it blurs the difference between objects and classes. Some authors do this deliberately and do not consider this a problem, like Beck and Cunningham (1989) in their original CRC paper. However, Beck and Cunningham worked with skilled professionals and not novices. In particular the roleplay phase is a major source for class/object confusion, since the CRC-cards are used as object surrogates. “You can pick up a card and talk about it as if it were the object itself, forgetting that the card ‘stands in’ for a ‘real’ object” (Wirfs-Brock & McKean, 2003 p99). However, this is technically not true. Different objects will likely have different “actual knowledge,” which in turn might affect an object’s behaviour. Two book objects for example will have different authors, titles, return dates, etc. To avoid object misconceptions it is furthermore recommended to use scenarios that feature several instances of at least one class (Holland, Griffiths & Rugaber, 2002). This makes the differences between the concepts of object and class more apparent. Biddle et al. (2002) also noted that students have problems to distinguish objects and classes. They suggest “to increasingly say that the cards represented objects, not classes ... [and] found this helped in later roleplay.” We suggest to treat CRC-cards as classes only. Letting them act as substitutes for objects during the roleplay is a first step towards class/object confusion. Instead, we propose the usage of specific “object cards” that act as instances of the CRC-cards.

Another problem is that students have difficulties staying on the right level of abstraction. Although abstraction is considered fundamental for software development, this seems to be a general problem for novices (Frorer et al., 1997; Bucci et al., 2001). Roleplaying is about testing responsibility distributions, not about testing implementation details. However, it is quite difficult to decide in general how to distinguish a design decision (responsibility distribution) from an implementation detail. Students need to think and argue at the right level of abstraction. While those with (even small) programming knowledge tend to dive into implementation details, those without previous knowledge may skip over interesting lower-level responsibilities. They likely have an anthropomorphic understanding of objects conceptualising them as ‘intelligent beings’ assuming that it is not necessary to specify certain details explicitly. According to Ben-Ari (2001) this may be due to a lack of a conceptual model of a computer. Here the teacher has to constantly adjust students thinking to the adequate level of abstraction, which is a difficult issue during a roleplaying session.

A related problem is that students easily get lost in tracing the model. The problem increases with the complexity of responsibilities and the number of objects involved in a scenario. Distribution of responsibilities increases communication between objects. Roleplay participants have to send requests to other objects, thereby eventually passing additional information. In a roleplay session a student might therefore be tempted to serve a request by means of the responsibility of a single object, disregarding the necessity to collaborate with other objects due to the request’s inherent complexity. In our example from section 2, books have a responsibility `check out` that does not have any collaborators (see figure 1). So, students might assume that checking out a book simply means to first check whether the book is on loan and if not, simply register it as checked out, as suggested in figure 2. However, the purpose of roleplaying scenarios is to evaluate a CRC-card model, which will likely be far from complete and stable, in particular in the beginning. Actually, checking out a book would at least require that the book arranges for a proper return date and registers with its borrower. Apart from doing some local work (like checking whether it is on loan and remembering the return date), the book would need to send requests to certain other objects in some order to get its ‘job’ done.

Another problem is buried in the approach itself. The CRC-card approach anthropomorphizes objects to facilitate understanding and to build a bridge between previous knowledge and the object oriented domain. On the other hand, we observed two different problems caused by anthropomorphizing objects. First, students often do not strictly rely on the local knowledge of objects and instead try to fulfil requests in co-operation with objects actually unknown to them or by reading their cards instead of explicitly sending a request. Second, students often fulfil responsibilities by magic, assuming their objects are somehow intelligent instead of strictly relying on the properties as written on the corresponding CRC-cards or object cards.

One possible origin of such pitfalls is due to the fact that the approach aims at **three different areas of understanding**:

1. *Understanding the approach*. How to proceed in the actual roleplay: What are the rules, what exactly do the object cards stand for and what to note on the object-cards?
2. *Develop a conceptual model of the object-oriented paradigm*. Understand how the CRC-card approach fits with the object-oriented concepts: What exactly are classes and objects and how exactly does an object-oriented program 'work.'
3. *Learn to model*. Understand the problem domain, understand the model, develop a 'feeling' for the quality of a model and understand the relationship between the static class model (on the CRC-cards) and the dynamic object interactions.

For a successful teaching concept, we believe it is crucial to be aware of the intended areas of understanding. Some of the problems known from using CRC-cards for teaching novices stem from mixing these areas. This blurs the differences between introducing, understanding, and using tools and concepts. In figure 2, we summarise different aspects of the learning process by distinguishing the use of the example on the one side (vertical) and the different comprehension levels on the other side (horizontal).

Vertical (example oriented) / **Horizontal (conceptual level)**	***General understanding* of *object-orientation* (OO)**	***Understanding* a given *example***	***Modelling* and *Testing* an *example***
Understanding OO *concepts*	Objects and classes; methods and attributes; associations; encapsu-lation ...	Actual local responsibilities that can only be discharged to specific, predefined collaborators	Sending of messages between actual objects to accomplish complex tasks
Learning a systematic approach for *modelling*	Responsibilities and collabo-rations as concepts; CRC-cards and RPDs as notations	Rules for finding classes, distribution of responsibilities	Develop concrete scenarios for several use case; modify/extend model as necessary
Learning a method for *testing* OO-model	Rules for the actual roleplay; objects rely on local knowledge	Acting out given scenarios using object's current properties; docu-ment a roleplay in a RPD	Roleplay as a tool for software comprehension

Figure 2: Matrix of Objectives, according to conceptual level and didactical use of example.

The success of the CRC-card approach relies on suitable example problems to be modelled (figure 2, third column) and example models to be explored (figure 2, second column). It seems reasonable to progress from exploring a given example to the construction of models, thereby following Bloom's taxonomy of learning objectives: From comprehension of concepts, over applying, analysing, synthesising to evaluation (Bloom 1956). According to Bloom, these levels rely on each other, so that a student can understand a level only if the preceding levels are comprehended.

4. A SYSTEMATIC INTRODUCTION OF THE CRC-CARD APPROACH

In order to avoid describing teaching recipes only, we reconstruct our teaching rationales using a theoretical grounding. This should help to make the internal coherence of the different elements of the teaching approach more feasible. This should reduce the risk of making decisions, which are locally 'correct' but are globally wrong, leading to lack of coherence in the teaching concept. Before that, we introduce the idea of roleplay diagrams, to be used in our approach to CRC-cards.

4.1. Roleplay Diagrams

Roleplay Diagrams (RPDs) are used to document object interaction. The objects in a RPD are instances of the classes modelled by CRC-cards. RPDs cover the most important aspects from UML object and communication diagrams (Booch, Rumbaugh & Jacobson, 1999). RPDs are however simpler and more informal and therefore better suited for recording scenarios in parallel to the roleplay activities.

Objects in RPDs are described by object cards. An object card is an instance of a CRC-card showing the instance's name, class name and current knowledge as shown in figure 3. All information presented on an object card must conform to the responsibilities on its corresponding CRC-card. However, we usually only list those properties that are relevant for the current roleplay activity. Large post-it notes work fine for this purpose. For each new object, we create a new object card and put it on a white-board or large sheet of paper. An object card is initialised with the knowledge for a specific object/instance, using the corresponding CRC-card as a template. In a library system, we might for example have an object card for object aBook representing the book with title 1984 (see figure 3).

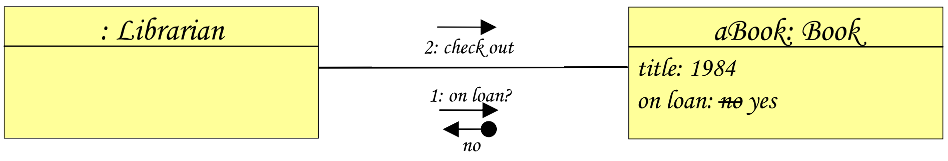

Figure 3: An excerpt from a roleplay diagram for checking out a book from a library.

Objects that "know" each other are connected by a line. During the roleplay, communication between objects is only possible if their corresponding object cards are connected. An object card can only be connected to the collaborators listed on its corresponding CRC-card. New objects can always be created, if their corresponding CRC-card is listed as a collaborator of the creator.

Actual communication is documented in the form `number: request`. `Number` keeps track of the ordering of requests and `request` corresponds to the actual service requested. A small arrow denotes the direction of the request. Requests can furthermore be annotated with data/object flows to explicitly document information that is sent or returned.

RPDs must always be consistent with the corresponding CRC-card model. Requests to and knowledge on an object card must conform to the responsibilities on its corresponding CRC-card. In our example in figure 3, we assume that class Book has responsibilities to handle the `on loan` and `check out` requests (which are the case according to the `Book` CRC-card in figure 1). Information on the object cards is updated to reflect state changes.

At the end of a scenario roleplay, the RPD provides an exact documentation of what happened, including the state changes. If necessary, this information can easily be translated into for example UML sequence diagrams.

4.2. Background for a Coherent Teaching Concept

We have chosen the Cognitive Apprenticeship (CA) approach, because it has proven to be a useful approach in different subject areas (Collins, Brown & Newman, 1989) as well as in teaching object technology (Tholander, Rutz, Karlgren et al., 1999).

CA aims at describing the most relevant features of successful teaching concepts in a coherent way, so that different elements are mutually supported. The **main ideas** are: Because in cognitive areas experts' problem solving strategies are not observable in teaching and learning, the *thinking processes of experts have to be made visible*. Observed *tasks should be situated in authentic context*, in order to make their relevance understandable to the learner. Examples should be generalised; *situations should vary* to make transfer more likely.

CA unfolds these ideas for successful learning environments in **four dimensions** (see figure 4).

Dimension	**Aspects of the dimension**
knowledge types	Domain knowledge, heuristic strategies, control strategies, learning strategies
teaching methods	Modelling, coaching, scaffolding, articulation, reflection, exploration
learning sequences	Global before local skills, increasing complexity, increasing diversity
sociological aspects of learning	Situated learning, community of practise, intrinsic motivation, co-operation

Figure 4: Cognitive Apprenticeship: Dimensions of successful learning environments.

Applying these dimensions to the domain of teaching object technology to novices by means of CRC-Cards and RPDs, we identify the following **important aspects**:

1. Knowledge types: Additional to teach basic concepts, *students should learn to model* (heuristics) and have to be able to test and evaluate their (CRC-card) models; help them exploring models.
2. Teaching methods: *Use CRC-Cards and RPDS as scaffolds*; encourage articulation and reflection by analysing and evaluating CRC-Models; coach them in using CRC-cards and RPDs.
3. Learning sequences: *Provide students with a conceptual model*; let them only afterwards engage in design activities.
4. Sociological aspects: Use *small, but realistic examples* for which meaningful uses case scenarios can be found (situated learning).

According to the above mentioned different levels of understanding, we suggest to use a learning sequence in which the learners attention in a first step is drawn to a vague understanding of object-orientation (a conceptual model). Understanding should include the according notations and their use. In a second step then, students grasp the conceptual model, so that they can build, test and evaluate simple object oriented models, using the methods and notations being taught to them.

The *first step* intends to conceptualise an overview of object-oriented technology, introducing the expert language and suitable tools. Novices explore a given problem domain and the associated CRC-model by means of roleplay. Thereby the following concepts and tools are introduced: class, object, association, method, attribute, object interactions, object structures, CRC-cards, RPDs and roleplays. In the *second step* this knowledge is used to introduce more specific concepts and to develop modelling skills. Note that although the learning sequence follows Bloom's taxonomy knowledge and skills are bound to situations in which they are learned. According to CA, actual constructivist and situated approaches to learning, learning occurs rather as a by-product of acting in situations. In step two comprehension of concepts is reinforced and substantially deepened, although it is explicitly targeted in step one. Therefore the two steps should be seen as a help for teachers to organise a learning sequence according to CA: with fading teacher support and increasing independence of students.

4.3. Step One: Learning to Think in Object Structures

The aim of this step is to build a bridge between the knowledge of novices and the (abstract) concepts of object technology by providing learners with a conceptual model, which focuses on objects, classes, responsibilities and collaboration. This is done by using the informal notation of CRC-cards, a given example from a problem domain familiar to the students (like the library-example mentioned above) and by anthropomorphizing objects.

CRC-Cards should "immerse the learners into the objectness of the material" (Beck & Cunningham, 1989) by providing an informal tool which concentrates object technology to classes, responsibilities and collaboration. A CRC-Model is a set of cards, describing a problem domain from an object-oriented viewpoint. The first example given to the students serves as a basis, from which conclusions and generalisations are drawn. Especially to programming novices the first example presented serves as a in CA so-called model from which generalisations are drawn. These form students conception how an object oriented program 'works'. Fleury (2000) reported a number of false assumptions of students about Java, all of them due to over-generalising the examples taught. In short, the first example serves as a conceptual model of an object-oriented program (Ben-Ari, 2001) and should be carefully chosen.

To avoid object misconceptions and enable a useful roleplay, the example should conform to certain standards (Holland et al., 1997; Fleury, 2000):

- The model should comprise several classes (at least three). From at least one class, there should be several instances (objects) to avoid class/object confusion (Holland et al., 1997).
- Objects should be 'active,' not only data-containers, to avoid the assumption that all objects are simple data records (Holland et al., 1997, p. 132) and to enable a rich roleplay.
- Responsibilities (i.e. methods) should be fulfilled by using collaboration to avoid the assumption that all methods work by using assignments, thereby fostering a procedural way of thinking (Holland et al., 1997, p. 132).
- Examples should rely on familiar problem domains, so that learners have some knowledge about the objects involved and can concentrate on modelling, i.e. the object-oriented 'viewpoint' of the domain.
- In order to be meaningful, the model should fulfil a realistic task in the problem domain.

Subsequently, a roleplay is accomplished, in order to clarify the functionality of an object-oriented program. In order to avoid getting lost in the game, RPDs are used to visualise and keep track of the state of the involved objects as well as the state of the overall object structure and the 'collaboration-history.' The diagrams help to replay the game, distinguish different versions and thereby encourage articulating the reasoning behind their choices. This will help them make their own conscious choices in later modelling activities and to develop an appropriate mental model of the dynamics of an object-oriented program. By fostering articulation and reflection (one of the core essentials in CA) students develop their own problem solving strategies based on viable mental models. The teacher can and should use this opportunity to 'listen to students' thoughts' about some import issues of an object oriented program (Fleury, 2000, p. 200). Moreover, by playing the game in front of the class, other students can share these thoughts.

The teacher must master the thin line between correctness and smoothness of the game. We suggest analysing the game in order to draw conclusions together in class. This helps to build an appropriate vocabulary, to establish a common shared model of object technology and to recalibrate false associations. Thereby the RPD can help and support an abstracted replay (Collins, Brown & Holum, 1991) of the game, in which the critical parts of the game are highlighted.

Anthropomorphizing objects can be a source for misconceptions, we therefore suggest to explicitly distinguishing between 'real life objects' and their abilities and those of objects in a computer. This can be done in combination with semi-formal definitions of concepts in a class discussion about the things learned through roleplay. These concepts are: class, object, state, object interactions, object structures, CRC-card, RPD and roleplay. It is important to stress that objects act only according to their state and the responsibilities listed on their corresponding CRC-card (i.e. their class). These analysing and reflection steps train students also in abstract thinking and in using formal notations.

A useful next step is to show real objects in a computer, using a graphical debugger as is for example provided by the BlueJ (Kölling, Quig, Patterson et al., 2003) or Fujaba (Schulte et al., 2003) environments. Thereby students can formalise the concepts object, state and collaborations a step further and are introduced in using tools, also.

In order to train modelling activities, a necessary next step is to use the approach in a more authentic setting: as a means to test and improve one's own model.

4.4. Step Two: Systematic Modelling and Testing Using CRC-Cards and RPDs

When students are accustomed to CRC-Cards, roleplays and RPDs these tools can be used as an approach to learn systematic modelling and testing of object-oriented models. According to Cognitive Apprenticeship therefore tasks should be placed in authentic contexts, enabling students to use concepts, tools and to engage in problem solving activities that are near to expert's style of working. This should help them to learn to think like experts and to pick up that tangible knowledge which is hardly found in textbooks but essential. Also, through increasing diversity and complexity of tasks learned skills are more likely to be applicable in solving real problems. So we suggest giving students a task to model, using CRC-cards and RPDs.

The CRC-card approach is especially well suited, when the problem is not well defined. During analysis and modelling the problem and application domains are analysed to understand the problem at hand. We identify the things to be done by the system to be developed and define classes (CRC-cards) for the actual system objects that accomplish these things. The set of CRC-cards constitutes an initial model of the system to be developed that can be validated by means of roleplaying. To support this process we have defined a step-by-step guide to the CRC-card approach (Börstler, 2004) (see figure 5).

- *find candidate classes* by means of brainstorming;
- *filter* the list of *candidates*;
- *create CRC-cards* for the remaining candidates;
- *allocate responsibilities* to CRC-cards/classes;
- *define scenarios* to test/evaluate our model;
- *prepare the group session*; *'roleplay' scenarios* using CRC-cards; *record scenarios*; and
- *update* CRC-cards and scenarios to reflect your findings.

Figure 5: Steps in our CRC-card approach to object-oriented modelling.

Please note that these steps are not performed in strict sequence. When filtering the list of candidates we usually have to consider possible responsibilities to make informed decisions. The last three steps are always performed in parallel.

The steps in figure 5 serve as a kind of cue (Collins et al., 1991), helping students in planning processes. Results from a study in the area of writing indicate that such procedural cues increased the time spend for planning, lead to more reflective activities and "the texts of experimental-group students were judged significantly superior in thought content" (Collins et al., 1991). These results may be applicable in the domain of object-oriented modelling. According to CA cues are helpful, because they provide students with an idea about experts' problem-solving strategies, including goal setting, generating ideas, revising and elaborating ideas, and so on. Cues should demonstrate how to use control strategies instead of giving recipes, thereby fostering independence activities of the students.

Students learn to use notations as tools for modelling, based on their idea about object technology, as learned in step one. Here, their ideas should be deepened by using more complex examples. For increasing diversity the task or problem domain should vary (compared to the example used in step one). To increase complexity the task could for example involve more complex scenarios involving more objects with more substantial responsibilities. Furthermore, the model could be implemented, thereby introducing programming language aspects. However, for the first example to be modelled by the students we suggest staying close to the original problem domain and complexity of the example used in step one. Learners should focus on using CRC-cards and RPDs to model, test and evaluate the model. In order to do so, the task should not be too complicated.

By exploring activities and problem solving strategies fairly close to 'real use' in software development, students learn to organise their thinking processes by means of the given tools (concepts, terminology, notations and processes). They learn to construct and validate an object-oriented model in a systematic way.

We strongly recommend students to work in groups, thereby fostering chances for verbalising thoughts, reflect on processes and results; and by comparing solutions of different groups learn to evaluate models. These processes include the proper use of technical terminology.

5. CONCLUSIONS

In this paper, we have evaluated object-oriented modelling using the CRC-card approach from a pedagogical point of view. Based on this evaluation and our experiences from using the approach **we recommend the following:**

1. *Be aware of the two major learning goals* of the approach; as a tool to introduce object-oriented concepts and foster object-oriented thinking and as a modelling tool to develop actually using CRC-cards as a modelling tool to systematically construct and validate object-oriented models in a systematic way.
2. *Introduce roleplay diagrams* to avoid the problems associated with using CRC-cards as object surrogates during the scenario roleplay.

The benefits we describe can however only be achieved when the teacher manages to keep focus on the particular main objectives as described in section 3. We propose a teaching concept designed to help teachers (and, of course, learners) to do so. Most learning goals are targeted in both steps of our two step approach described in section 4. However, in each step there is a clear focus on certain main objectives. Keeping these in mind prevents teachers from being side-tracked by seemingly import issues that might be irrelevant for the current learning goals. The main objective of step one is for students to acquire a valid conceptual model for object-oriented thinking. In step two, the main objective is for students to develop modelling skills.

Although the suggested items of the teaching concept as well as the general approach itself are grounded in pedagogical theory, our claims are—despite our own positive experiences in teaching according to this concept—to be seen as hypotheses, waiting to be evaluated by means of empirical studies.

It is important to recognise that learning and teaching object-oriented programming is not a programming language issue. The object-oriented paradigm requires a different way of thinking. Students (and teachers as well) need a valid and shared conceptual model for reasoning about object-oriented programs. The CRC-card approach can provide such a model. It is important to understand the close relations between the underlying concepts on one hand and the used methods, languages and tools on the other hand. The two step approach facilitates awareness of such issues. Novices need to be introduced to the manifold items and their relations without being overwhelmed, or forced to make false generalisations. We must help them to grasp the 'big picture' and not only isolated 'knowledge chunks' without obvious interrelationships.

So-called problem-solving capabilities are particularly important: Structuring, abstraction and formalisation; planning and revising. Nevertheless, we need to find guidelines or even standards for teaching and learning, indicating what competencies students should have after CS1.

REFERENCES

Andrianoff, SK & Levine DB 2002. Role playing in an object-oriented world. *Proceedings SIGCSE'02*: 121–125.

Beck, K 1993. CRC: Finding objects the easy way. *Object Magazine* 3(4): 42–44.

Beck, K & Cunningham W 1989. A laboratory for teaching object-oriented thinking. *Proceedings OOPSLA'89*: 1–6.

Bellin, D & Simone, SS 1997. *The CRC Card Book*. Addison-Wesley, Reading.

Ben-Ari, M (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching* 20(1): 45-73.

Biddle, R, Noble, J & Tempero, E 2002. Reflections on CRC cards and OO design. *Proceedings Tools Pacific'02*: 201–205.

Bloom, BS & Krathwohl, DR (Eds.) 1956. Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain. Longmans, Green, New York.

Booch, G, Rumbaugh, J & Jacobson, I 1999. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading.

Börstler, J 2004. Object-oriented analysis and design through scenario roleplay. Technical Report UMINF-04.04, Dept. of Computing Science, Umeå University.

Börstler, J, Johansson, T & Nordström 2002. Introducing oo concepts with crc cards and Bluej-a case study. *Proceedings FIE'02*: T2G–1–T2G–6.

Bucci, P., Long, T.J. & Weide, B.W. 2001. Do We Really Teach Abstraction? *Proceedings SIGCSE'01*: 26-30.

Clancey, M 2004. Misconceptions and attitudes that infere with learning to program. In Fincher, S & Petre, M (Eds), *Computer Science Education Research*: 85–100. Taylor & Francis, Lisse, The Netherlands.

Collins, A, Brown, JS & Holum, A 1991. Cognitive Apprenticeship: Making Thinking Visible. *American Educator*. http://www.21learn.org/arch/articles/brown_seely.html

Collins, A, Brown, JS & Newman, SE 1989. Cognitive apprenticeship: Teaching the craft of reading, writing and matematics. In Resnick, LB (Ed.), Knowing, learning and instruction: Essays in honor of Robert Glaser: 453-494. Erlbaum, Hillsdale, NJ.

Fleury, AE 2000. Programming in Java: Student-constructed rules. *Proceedings SIGCSE'00*: 197–201.

Frorer, P., Hazzan, O. & Manes, M. 1997. Revealing the Faces of Abstraction. *Computers for Mathematical Learning* 2: 217-228.

Go, K & Carroll, JM 2004. The blind men and the elephant: Views of scenario-based system design. *Interactions* 11(6): 44–53.

Gray, KA, Guzdial, M & Rugaber S 2002. Extending CRC cards into a complete design process. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, GA. http://www.cc.gatech.edu/ectropic/papers/.

Holland, S, Griffiths, R & Woodman, M 1997. Avoiding object misconceptions. *Proceedings SIGCSE'97*: 131–134.

Hvam, L, Riis, J & Hansen, BL 2003. CRC cards for product modelling. *Computers in Industry* 50(1): 57–70.

Kölling, M, Quig, B, Patterson, A & Rosenberg, J 2003. The BlueJ System and its Pedagogy. *Computer Science Education* 13(4): 249-268.

Otero, MC & Dolado JJ 2002. An initial experimental assessment of the dynamic modelling in uml. *Empirical Software Engineering* 7(1): 27–47.

Schulte, C, Magenheim, J, Niere, J & Schäfer, W 2003. Thinking in Objects and their Collaboration: Introducing Object-Oriented Technology. *Computer Science Education* 13(4): 269–288.

Sims-Knight, JE, & Upchurch, RL 1993. Teaching Object-oriented Design Without Programming: A Progress Report. *Computer Science Education* 4: 135-156.

Tholander, J, Rutz, F, Karlgren, K, & Ramberg, R 1999. Design and Evaluation of an Apprenticeship setting for Learning Object-Oriented Modeling. *Proceedings International Conference on Computers in Education*.

Teif, M & Hazzan, O 2004. Junior high school students' perceptions of object oriented concepts. *Eigth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts*. http://www.cs.umu.se/˜jubo/Meetings/ECOOP04/Submissions/TeifHazzan.pdf.

Wilkinson, N 1995. *Using CRC Cards, An Informal Approach to Object-Oriented Development*. SIGS, New York.

Wilkinson, N 1996. The role of informal techniques. *Journal of Object-Oriented Programming* 9(6): 28–32.

Wirfs-Brock, R & McKean, A 2003. *Object Design–Roles, Responsibilities, and Collaborations*. Addison-Wesley, Boston.

Wirfs-Brock, R, Wilkerson, B & Wiener, L 1990. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, NJ.