Using Conceptual Structures in the Design of Computer-based Assessment Software

Uta Priss, Nils Jensen, Oliver Rod

Zentrum für erfolgreiches Lehren und Lernen
Ostfalia University of Applied Sciences
Wolfenbüttel, Germany
www.upriss.org.uk, {n.jensen,ol.rod}@ostfalia.de

Abstract. This paper discusses the use of conceptual structures in the design of computer-based assessment (CBA) tools for e-assessment of programming exercises. In STEM (science, technology, engineering and maths) subjects, universities often observe high dropout and failure rates among the first year students. There are a number of research initiatives that investigate the use of interactive teaching methods and e-learning technologies for improving STEM education. This paper presents a conceptual model of programming exercises and discusses more generally how conceptual structures can be employed for the implementation of CBA tools.

1 Introduction

STEM (science, technology, engineering and maths) subjects are notoriously difficult to learn and teach as demonstrated by high dropout and failure rates among first year university students. There are a number of reasons for the difficulty of such subjects. Researchers in Physics Education Research (for example, Hestenes et al. (1992)) have observed that students often have misconceptions which are not easily overcome by traditional lecturing methods even if these include exercises and demonstrations. Hestenes et al. (1992) explain that misconceptions are commonsense beliefs which can be regarded as reasonable hypotheses grounded in everyday experience. Unfortunately, commonsense belief are not always correct. For example, Newtonian physics includes many concepts that are contradictory to commonsense beliefs and in fact counter-intuitive. Students find it very difficult to overcome such misconceptions. Even though they may be able to apply Newtonian concepts in calculations by following an algorithm (which is frequently sufficient for passing exams), if asked to provide conceptual explanations students will often revert to non-Newtonian, incorrect concepts. Furthermore, if students are passing an exam only by applying memorised facts and algorithms, they will forget the subject matter quickly after the end of the semester. On the other hand, as soon as students achieve a conceptual understanding of a subject matter they will often retain such knowledge for 20 years and more (Conway et al., 1992).

A second problem is what can be called a "teacher's dilemma". McDermott (2001) observes that at least in USA, the people who teach physics are not at all like the undergraduate students they teach because teachers have achieved a masters or doctoral

level of understanding of the subjects whereas undergraduate students have often no ambition or interest to progress in the subject any further than required. By definition, teachers are usually not people who have ever dropped out of university or experienced learning difficulties but instead usually have been comfortable with the learning styles presented by traditional university teaching. Teacher training attempts to help prospective teachers develop an understanding of the students' conceptual models. For example, Prediger (2010) discusses the diagnostic competences that maths teachers need to develop in order to be able to listen to students and to analyse and understand their thinking. Tall (1977) argues that learning of mathematics involves cognitive conflicts. The acquisition of new concepts by a maths student is not a continuous process but includes conceptual jumps and states of confusion and emotional upset. A teacher must be able to detect occurrences of conflict in the mind of a learner and select an appropriate approach for conflict resolution amongst many different possible approaches. A further potential challenge to be overcome are teachers' attitudes towards trying new teaching methods (Pundak et al. 2009). Interestingly, changing a teacher's pre-existing belief about teaching methods may not be any easier than it is for students to overcome their misconceptions.

While traditional lecturing styles seem to be less than optimal in STEM subjects, interactive engagement methods (Hake, 1998) appear to be more successful. Hake defines "interactive engagement methods as those designed at least in part to promote conceptual understanding through interactive engagement of students in heads-on (always) and hands-on (usually) activities which yield immediate feedback through discussion with peers and/or instructors". An example is Mazur's (1996) peer instruction which uses cycles consisting of questions that are voted on by the students, peer discussion, group discussion, debriefing and then again the original questions. Apparently, while students might find it difficult to learn from a teacher's explanations, they find it easier to understand complex concepts and resolve cognitive conflicts when they can discuss these with other students (peers) who tend to be at a similar level of conceptual development as they are. Thus peer instruction is a means of overcoming the teacher's dilemma. The teacher becomes more of a facilitator or coach than an authoritarian source of information. A theoretical foundation for this approach to teaching is a constructivist model of learning (e.g., Ben-Ari, 1998).

It would be of interest to replace the currently prevailing constructivist model of learning with a Peircean pragmatist model. Levy (2007) observes that Peirce already discussed a "teacher's dilemma" because "in order to learn you must desire to learn, and in so desiring not be satisfied with what you already incline to think" (CP.1.135)[1]. But a teacher needs to be reasonably convinced of the truthfulness of the subject matter to be able to teach. Thus the state of teaching (a state of belief) is in contrast with the state of learning which is a state of doubt. According to Levy, Peirce's solution to the dilemma is that a teacher must be willing to learn while teaching and that the learning process must be a cooperation between teacher and student. Therefore, it can be argued that Peirce anticipated interactive engagement teaching. A more in depth analysis of the relevance of Peirce's work for education would be of interest (in particular with respect

[1] The usual manner of citing Peirce' papers is adopted where CP refers to the Collected Papers of Charles Sanders Peirce followed by volume and paragraph numbers.

to a pragmatic instead of a constructivist philosophy). While that is beyond this paper, the ICCS community might be a suitable audience for such research.

The core application area of this paper within STEM education is teaching programming to computer science students. In particular we are interested in how conceptual structures can be used to support tools for teaching programming such as computer based assessment (CBA) tools. With "conceptual structures" we are referring to tools and technologies commonly used in the ICCS community, for example, conceptual graphs and formal concept analysis. Section 2 introduces CBA tools. Section 3 describes a conceptual model of programming exercises. Section 4 discusses more generally how conceptual structures can be used to support CBA tools. The paper ends with a short concluding section.

2 Computer-based assessment software

A large body of literature exists on the topic of STEM education. Our particular interest is the teaching of programming languages in computing or similar formalisms in mathematics. In this domain, computer-based assessment (CBA) software has been developed which allows students to submit code that is automatically evaluated (e.g. Pears et al. (2007), Rongas et al. (2004)). CBA software is more narrow in scope than virtual learning environments or course management systems which usually provide access to lecture materials, timetables and communication tools. If virtual learning environments provide automatically evaluated assessments at all, these are of a simpler, more static nature such as multiple-choice or fill-in the blanks tests. CBA tools without graded assessments are quite popular as add-ons to on-line tutorials which contain pastebins for sourcecode execution[2]. Advantages of using CBA tools in university courses are according to Pears et al. (2007) the fact that even students in large classes can be provided with detailed feedback in a timely manner. Automatic assessment is often seen as more fair and objective than assessment by tutors. Since CBA tools are employed in practicals, not lectures, they are more likely to be used with interactive engagement methods. Drawbacks of CBA tools are that exercises need to be specified very carefully to avoid misunderstandings. Furthermore automatic evaluation can miss problems. A student's work could receive full marks although it is written poorly and contains errors that were not anticipated by the designers of the exercise. Unrestricted access to instant feedback can encourage students to employ a trial-and-error approach to programming.

CBA tools should be deployed with a suitable pedagogical method as can be found in the literature, for example, by Leron & Dubinsky (1995) since the 1990s. Without a sound pedagogical method or without being embedded into an interactive engagement style of teaching, CBA tools may not provide any benefits. If used correctly, CBA tools save time because the tools provide automated feedback and can be used by large numbers of students simultaneously - only limited by the size of the computer labs. Ideally the time lecturers save by not having to provide feedback on simple mistakes which are automatically detected by the CBA tool, lecturers should spend on helping students with conceptually challenging problems (or misconceptions) that require more in depth discussion (Priss et al. (2012b)).

[2] For example the Tryit editor at www.w3schools.com or the SQL tutorials at sqlzoo.net.

Creating exercises for a CBA tool is more labour-intensive than creating other exercises because CBA exercises need to be specified very precisely so that they cannot be misinterpreted by students and they need to be tested before being used. Furthermore, the algorithms used for automatically evaluating the student-submitted code need to be provided usually either using software testing methods or intelligent tutoring techniques. CBA tools are only labour-saving if tools are provided that assist lecturers in the creation of exercises and the exercises can be reused. Thus in addition to the software required for the CBA tools themselves, one needs authoring tools and an infrastructure for the storage, retrieval and exchange of exercises. These are the areas where we see conceptual structures as potentially very useful. Although there are already many existing e-learning tools and standards for exchanging exercises available, as Rey-Lopez et al. (2008) observe these existing tools are not suited for the more detailed and content-rich exercises used for teaching programming. The problem of exchanging programming exercises and integrating CBA tools with other e-learning tools is according to Rey-Lopez et al. still an unsolved problem.

3 A conceptual model of programming exercises

In order to improve authoring tools for CBA software and to support exchanging exercises across tools and users, a solid understanding of the conceptual structure of programming exercises is beneficial. This section discusses a conceptual model of programming exercises developed using the Protege[3] editor. The only reason for using Protege was because it has a sophisticated, stable user interface and many graphical output options. The functionality used was classes, is-a relations and attributes (or slots) with value restrictions which are provided by many kinds of conceptual structures tools. Thus the discussion in this section is not meant to focus on the technology used but instead on the conceptual model that was derived.

Figure 1 represents an overview of all of the classes of the conceptual model. Figure 2 shows different types of feedback. An exercise can be evaluated by any combination of automated tests, peer review from other students and feedback from lecturers. The distinctions are useful because each type of feedback has a different functionality in the system. For example, evaluations by automated tests and teachers contribute to the marking scheme. Feedback by students is sometimes considered a student-only affair which cannot be viewed by the teachers. The attributes “line Number” and “location” are useful for the visual presentation of the feedback for the student. Automatically generated feedback usually has a precise location, that is a particular line of code which raised an error or a warning. Feedback that is written by other students or a lecturer can only be economically connected to a location, if authoring tools are used that allow to annotate code.

Resources as in Figure 3 tend to be provided as text or files. This distinction is of technical interest because text and files are implemented differently. For example, if students are asked to submit files, these need to be checked for file size and type. In the early stages of a programming class, students are often asked to write only parts of a

[3] http://protege.stanford.edu/

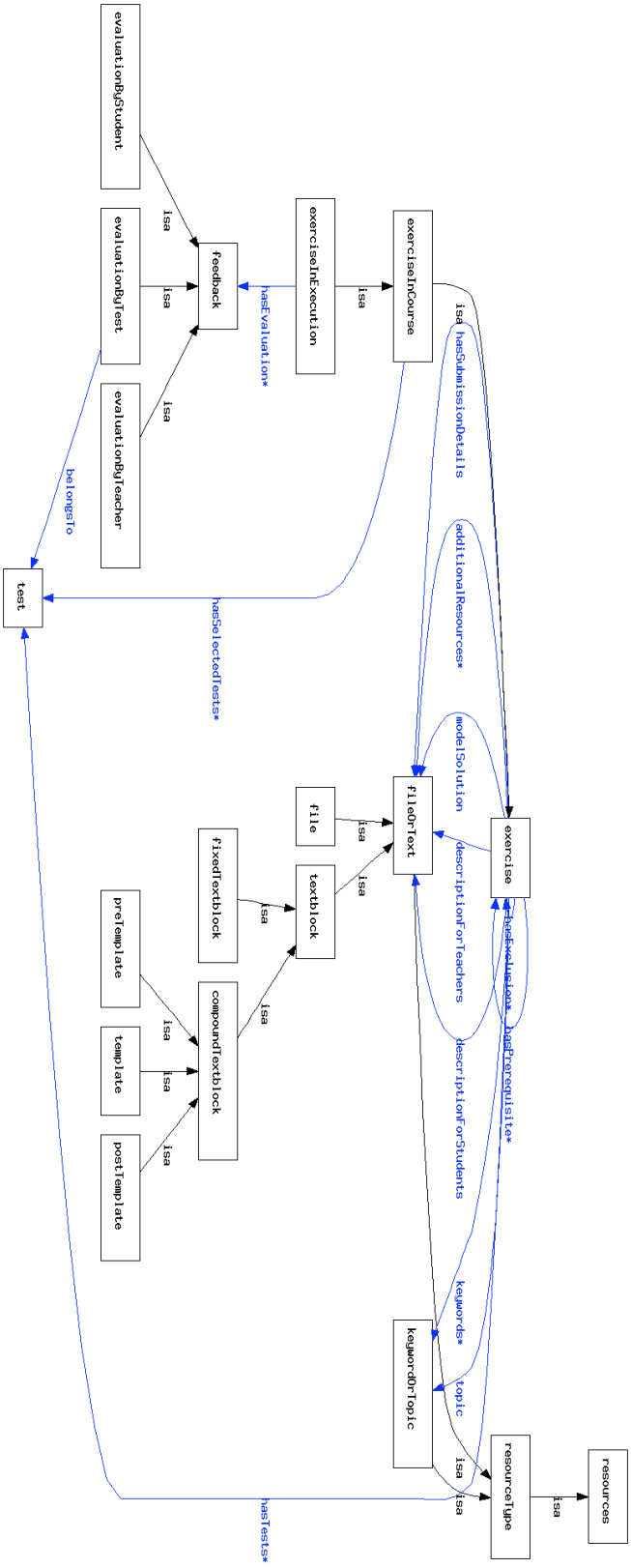

Fig. 1. Overview of the model

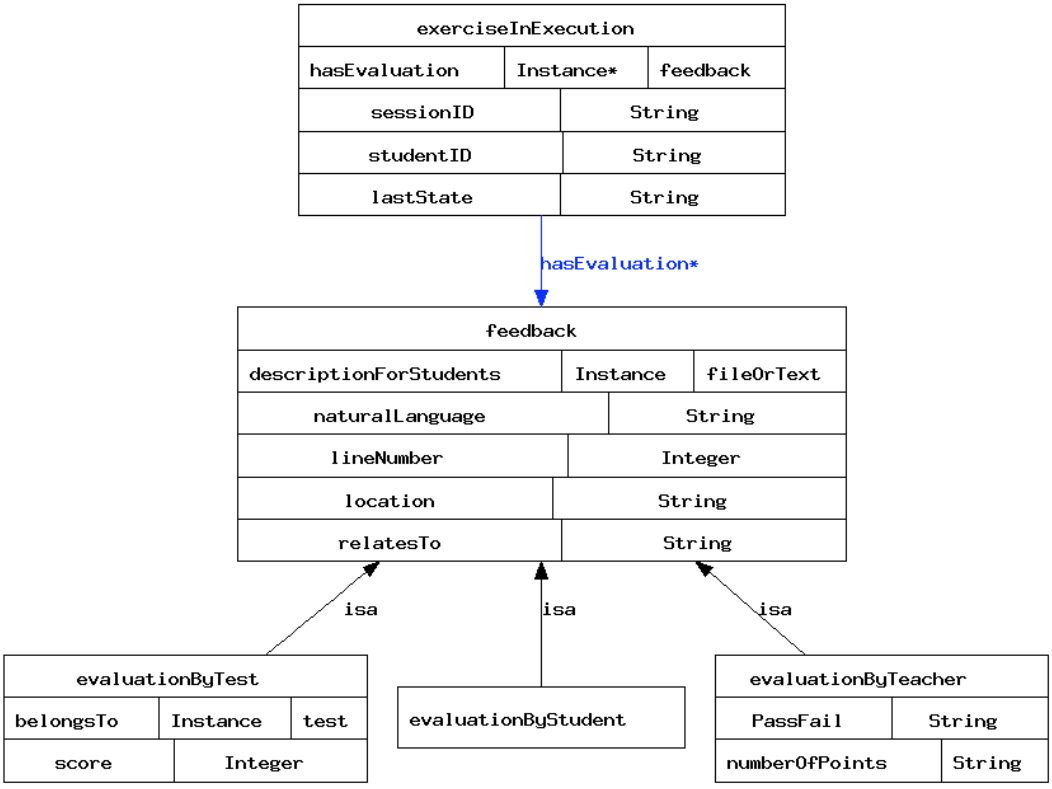

Fig. 2. Feedback for an exercise can be provided by test, peer (student) or teacher

program, for example, just a while loop. The CBA tool can either provide a template to the student which contains the code that the student is expected to modify or it can hide some code completely and automatically attach it before or after a student-submitted code snippet. Each resource has a "resource User" attribute which determines who has access to the resource, in particular whether the students are allowed to see the resource.

Figure 4 shows that a programming exercise exists on three levels: a general description independently of when and where the exercise is used; an "exercise in course" which has additional attributes about deadlines and about the actually selected tests from all available tests; and an "exercise in execution" which contains attributes about the student-submitted code, its evaluation results and session and state information so that a student can return to an exercise which has not been completed.

Exercises themselves are part of an ordered set: each exercise can have some prerequisites which the students need to pass beforehand. This is particularly useful if the CBA tool has intelligent tutor functionality. In order to avoid plagiarism, it is helpful to have a larger question bank from which randomised questions are selected so that not every student receives the same questions (Russell & Cummings, 2005). This means

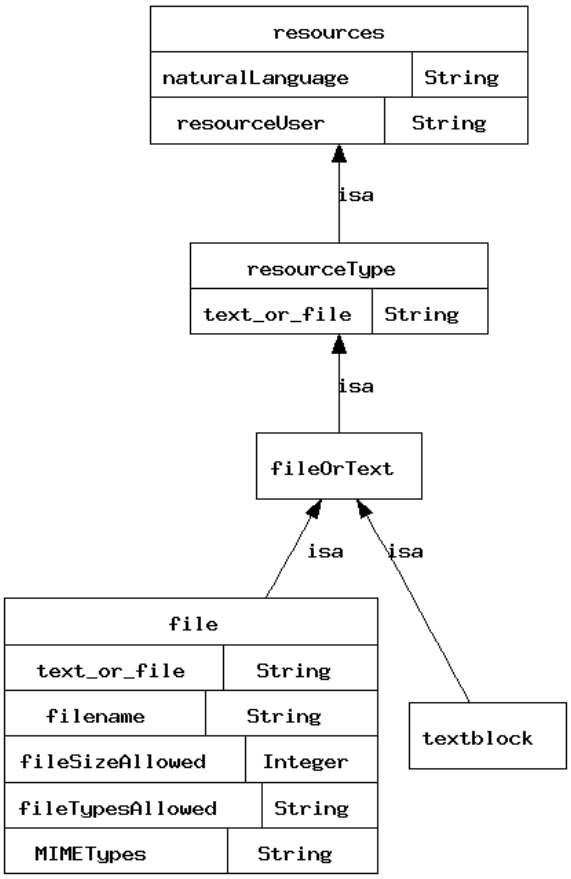

Fig. 3. Resources that can be up- or downloaded can be textblocks or files

that there needs to be an equivalence relation on the exercises (“has Exclusion”) which shows which exercises are of similar difficulty and content and can be used as alternatives.

Last but not least, Figure 5 provides examples of available tests. Many current CBA tools use standard software engineering tests (unit, style checking and code coverage tests) for evaluating student-submitted code. CBA tools often incorporate standard testing software for such purposes so that the lecturers need not learn new technologies for writing their tests. Some CBA tools support writing blackbox tests which analyse the in- and output of a program.

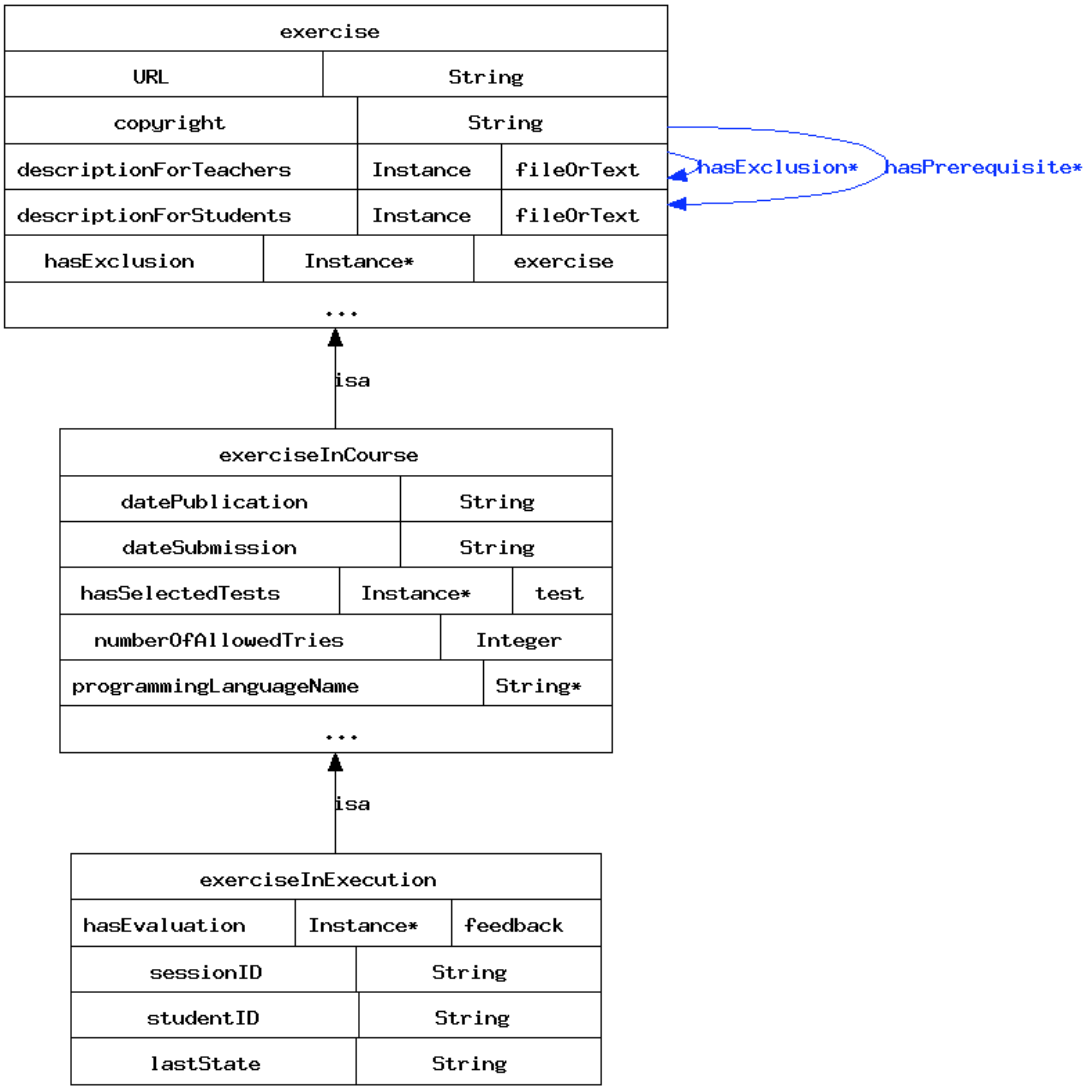

Fig. 4. An exercise exists on three levels: abstract, in a course or submitted by a student

4 How Conceptual Structures can help

This section provides an analysis of the different aspects involved in preparing and using exercises with CBA tools. Figure 6 shows a life cycle of CBA exercises. As explained in Section 2, in order to be cost-effective the additional cost required for creating exercises must be balanced by the benefit of reusing and sharing of exercises. Thus a community must be established that shares and reuses exercises. This community could

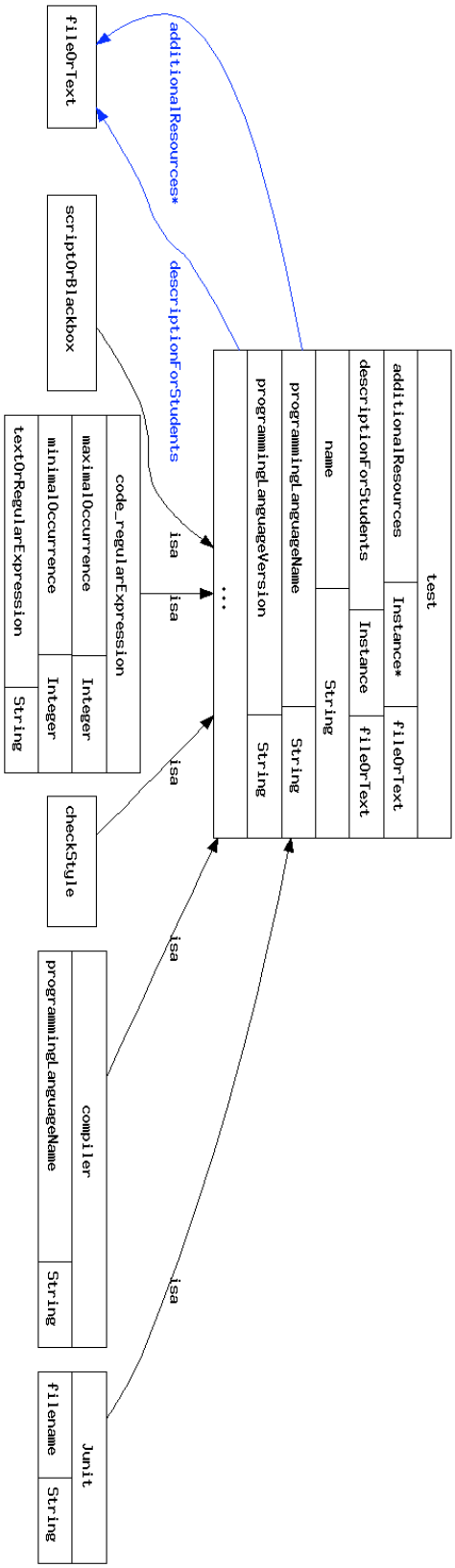

Fig. 5. Different types of tests

range from just a few lecturers within a department to lecturers in 100s of universities as, for example, the user group of the Lon-Capa[4] software. In order to share exercises, there must be a mechanism that allows lecturers to find appropriate exercises which can be a challenging task if there are large numbers of exercises available. The individual stages of the life cycle are discussed in the subsections below with reference to how the conceptual model developed in this paper can help.

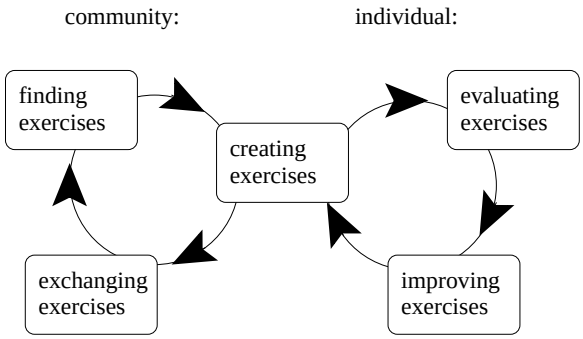

Fig. 6. The life cycle of CBA exercises

4.1 Searching and finding learning materials

Large amounts of learning materials already exist and are available for reuse. It is beyond this paper to review the literature on this topic in any detail. It might suffice to mention the open educational resource (OER)[5] efforts or the fact that Lon-Capa contains more than 200,000 resources (Kortemeyer, 2006). A major challenge in this area is how to build search tools that help lecturers to find relevant materials. Most likely there are already many duplicate or similar documents amongst the available materials simply because lecturers do not know what is available. General purpose search engines only retrieve the most popular documents which may not be the most relevant.

As an example, the Lon-Capa software provides metadata for its more than 100,000 exercises. Some of the metadata are created by the authors of the exercises. Some are dynamically generated, for example, information about in how many and which courses an exercise is used and what the student results are for each exercise. A review of the existing metadata recently revealed[6] that the manually created metadata are entirely useless for search purposes because they are inconsistent and often missing (supporting what has been known in the library and information science community for decades). In fact in the future, Lon-Capa may drastically reduce the collection of manually created metadata and focus on the automatically generated data instead.

[4] http://www.lon-capa.org
[5] http://en.wikipedia.org/wiki/Open_educational_resources
[6] G. Kortemeyer, personal communication, August 31, 2012

Currently most of the Lon-Capa exercises are of a more static nature and not programming exercises. We argue that the metadata of programming exercises according to the conceptual model in this paper is also manually created, but of a different type than the already existing metadata in Lon-Capa. The information recorded in the conceptual model is essential to the functioning of a programming exercise. For example, it is necessary to specify what programming language and what tests are to be used and how the marks are calculated. Once an exercise has been created this information is precise and unambiguous. This metadata is different from metadata, such as subject headings or keywords, which are more subjective, optional and debatable.

We argue that a conceptual model of programming exercises as developed in this paper can improve retrieval of programming exercises. In contrast to manually created metadata which tend to be inconsistent, our model structures only the essential data of the exercises which is thus rendered more accessible for searching. Other search details can be obtained from automatically collected metadata (such as the degree of difficulty of an exercise from the metadata about student results). Typical, re-occurring examples of programming exercises (such as the "Towers of Hanoi" or "Fibonacci numbers") can be found by searching within the full text of the exercises. Thus, an automatic exploitation of structured data, automatically generated metadata and the full text of the exercise with standard data mining methods is possible.

4.2 Exchanging exercises

In order for exercises to be exchanged, a standard format needs to be defined which represents the data and metadata of the exercise in a structured manner. An XML representation of the conceptual model developed in this paper could be an example of such a format. As mentioned before, Rey-Lopez et al. (2008) observe that existing standards for learning materials are not suitable for representing the greater amount of detail required for programming exercises. Automatically-assessed programming exercises not only need a description of the content of the exercise but also of the technical requirements, for example, as to which programming language, which versions of the tools, and which testing technologies are used. Furthermore, if the exercises are to be exchanged in a manner that does not require extensive amount of manual editing for importing exercises, then there need to be means for automatically detecting version differences and to convert into formats required by a specific tool.

There is currently an effort to create an exchange format for programming exercises undertaken by a working group as part of the eCULT project[7] which we are part of. Because that work is on-going and yet unpublished we cannot discuss more details about the format in this paper. Our contribution to the working group is based on the conceptual model developed in this paper. But because not all members of the working group are familiar with conceptual structures, the exchange format is represented in XML and not conceptually. Our conceptual structures model is somewhat more detailed and abstract than the format developed by the group and represents our view of the topic.

[7] http://www.ecult-niedersachsen.de/

4.3 Creating Exercises

Creating programming exercises consists of creating the content and the technical implementation. With respect to developing appropriate content, Priss et al. (2012a) discuss in detail how conceptual structures (in the form of Formal Concept Analysis (FCA)) can be used for modelling conceptual difficulties in learning processes in mathematics. A conceptual model as represented in this paper provides structures for authoring tools for programming exercises that can be used in implementations. Programming exercises need a detailed specification of testing tools, versions and the tests themselves. Some details are repetitive and could be supplied semi-automatically; other details are specific for each exercises and need to be manually supplied. In our experience it takes about 2 hours to convert an existing programming exercise into one that is usable with a CBA tool. Using the conceptual model developed in this paper, it would be possible to design templates that would shorten the time required for writing exercises.

4.4 Evaluating and improving exercises

As mentioned in Section 2, the quality and precision of CBA exercises must be higher than that of manually-assessed exercise. If a lecturer makes a mistake in the wording of a manually-assessed exercise, this mistake can be rectified when the exercise is marked, for example, by adjusting the marking scheme to reflect that slightly different interpretations of the exercise are acceptable. If a CBA exercise is ambiguously worded and thus provides misleading feedback to students then the labour-saving effect of the exercise is lost because the lecturer needs to contact every individual student to provide additional information to remove the ambiguity. The resulting confusion could easily destroy any pedagogical benefit of using a CBA tool. If the CBA tool is used for an exam, the exercise may need to be manually-assessed after all. If the error is detected too late, the whole assessment may become worthless; or if the error is not detected at all, students will receive unjustified marks. Therefore exercises need to be well-tested before they are used with larger groups of students. During and at the end of a semester, the performance of the exercises needs to be evaluated, for example, by statistical analysis of the points students achieved for each exercise. High failure rates for an exercise could indicate that there is a problem with the wording of the exercise or it could be that the exercise highlights a misconception which the students have that must be addressed by other learning materials. Based on the evaluation, exercises (and supporting learning materials) should then be improved before they are used again.

Evaluation and improvement of exercises can only be performed by individual lecturers who use the exercises in their course. But the improvements of an exercise then need to be shared again with the community. The Lon-Capa software has essentially solved these problems by establishing mechanisms for creating and maintaining copies of exercises that have been modified and for communicating changes to other current users of an exercise. Furthermore, Lon-Capa provides mechanisms for alerting authors of exercises to potential problems detected with an exercise. A conceptual model could further assist by providing additional semi-automated checks, for example, if the version of a programming language is changed for one exercises it could automatically be checked whether other exercises might require a similar change.

5 Conclusion

This paper argues that teaching STEM topics is difficult by nature but pedagogical methods and tools exist that lead to improved teaching success. With respect to teaching programming languages, CBA tools can be beneficial if they are employed with a suitable interactive engagement style of teaching. The creation, maintenance, exchange and retrieval of programming exercises is labour-intensive but can be supported by conceptual structures. An example of a conceptual model for programming exercise is presented in this paper. The model is currently being used to guide our involvement in a working group for creating an exchange format for programming exercises and as a design aid in our implementation of a CBA tool which is further described by Priss et al. (2012b).

Acknowledgements

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01PL11066H. The sole responsibility for the content of this paper lies with the authors. We would also like to thank the other members of the eCULT working group on creating an exchange format for programming exercises: Sebastian Becker, Stefan Bisitz, Helmar Gust, Sven Strickroth. We have been careful not to use any materials from those discussions in this paper but it is likely that the conceptual model developed in this paper has been influenced to some degree by discussions of that group.

References

1. Ben-Ari, Mordechai (1998). *Constructivism in computer science education.* SIGCSE Bull, 30, 1, p. 257-261
2. Conway, M. A., Cohen, G.; Stanhope, N. (1992). *Very long-term memory for knowledge acquired at school and university.* Applied Cognitive Psychology, 6, p. 467-482.
3. Hake, Richard R. (1998) *Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses.* American Journal of Physics, 66, 1, p. 64-74.
4. Hestenes, D.; Wells, M.; Swackhamer, G. (1992) *Force Concept Inventory.* Phys. Teach., 30, p. 141-158.
5. Kortemeyer, Gerd (2006). *The Evolving Growth of LON-CAPA*. Campus Technology, 10/03/06. Available at http://campustechnology.com/articles/2006/10/the-evolving-growth-of-loncapa.aspx.
6. Leron, Uri; Dubinsky, Ed (1995). *An Abstract Algebra Story.* The American Mathematical Monthly, 102, 3, p. 227-242.
7. Levy, Ronald (2007). *Peirce's Theory of Learning.* Educational Theory, 2, p. 151-176.
8. Mazur, Eric (1996). *Peer Instruction: A User's Manual*. New Jersey, Prentice Hall.
9. McDermott, Lillian Christie (2001). *Oersted Medal Lecture 2001: "Physics Education Research-The Key to Student Learning".* American Journal of Physics, 69, 11, p. 1127-1137.
10. Pears, Arnold; Seidman, Stephen; Malmi, Lauri; Mannila, Linda; Adams, Elizabeth; Bennedsen, Jens; Devlin, Marie; Paterson, James (2007). *A Survey of Literature on the Teaching of Introductory Programming.* SIGCSE Bull., 39, 4, p. 204-223.

11. Prediger, Susanne (2010). *How to develop mathematics-for-teaching and for understanding: the case of meanings of the equal sign.* J. Math. Teacher Educ., 13, p. 73-93.
12. Priss, Uta; Riegler, Peter; Jensen, Nils (2012a). *Using FCA for Modelling Conceptual Difficulties in Learning Processes.* In: Domenach; Ignatov; Poelmans (eds.), Contributions to the 10th International Conference on Formal Concept Analysis (ICFCA 2012), p. 161-173.
13. Priss, Uta; Jensen, Nils; Rod, Oliver (2012b). *Software for E-Assessment of Programming Exercises.* In: Goltz et al. (eds.), Informatik 2012, Proceedings of the 42. Jahrestagung der Gesellschaft fr Informatik, GI-Edition, Lecture Notes in Informatics, P-208, p. 1786-1791.
14. Pundak, David; Herscovitz, Orit; Shacham, Miri; Wiser-Biton, Rivka (2009). *Instructors' Attitudes toward Active Learning.* Interdisciplinary Journal of E-Learning and Learning Objects, 5, p. 215-232.
15. Rey-Lopez, M., Brusilovsky, P., Meccawy, M., Diaz-Redondo, R., Fernandez-Vilas, A.; Ashman, H. (2008). *Resolving the Problem of Intelligent Learning Content in Learning Management Systems.* International Journal on E-Learning, 7, 3, p. 363-381.
16. Rongas, T.; Kaarna, A.; Kalviainen, H. (2004). *Classification of Computerized Learning Tools for Introductory Programming Courses: Learning Approach.* In Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04), IEEE Computer Society, p. 678-680.
17. Russell, G.; Cummings, A. (2005). *Online Assessment and Checking of SQL: Detecting and Preventing Plagiarism.* In: 3rd Workshop on Teaching Learning and Assessment in Databases (TLAD 2005). HEA-ICS, p. 46-50.
18. David Tall (1977). *Cognitive Conflict and the Learning of Mathematics.* First Conference of The International Group for the Psychology of Mathematics Education at Utrecht, Netherlands.