

Creating, Filling, and Using a Repository of Reusable Learning Objects for Database Courses¹

MICHAEL KLEIN, KHALDOUN ATEYEH, BIRGITTA KÖNIG-RIES, JUTTA MÜLLE²

1 INTRODUCTION

When you take a look at courses on database management systems taught by different instructors at different types of universities for different target audiences, you will notice that there are quite a number of topics that are part of nearly every (or at least a large number of) courses. These topics include, to name but a few, database design methods, query languages, transaction management and so on. The production of high-quality learning materials is a time consuming and expensive process. Nevertheless, it is done over and over again for these topics - more or less anew for each new course. Reuse of material happens sometimes by its author, but even that is rather scarce. Why is it that material is not reused more often? One reason of course are copyright restrictions, however, more often than not, authors are more than willing to allow others to reuse their material. In our experience, gained within the Vikar project [14], the main obstacle to widespread reuse of learning material is the lack of modularization of the existing material. Courses are large, monolithic entities, which makes it hard to adjust them to differing needs. Learning material should be offered for reuse in small, well described building blocks that can be freely combined. Our SCORE (System for Courseware Reuse) system [11] is a platform that allows for the generation of such a repository of modularized learning materials and supports the usage of these materials to generate new courses. In this paper, we will show how the repository is created, how existing material can be fed into it, and how a new course can be build using the existing and new materials. As an example, we have chosen one of the topics that occur frequently in database courses, normalization. The remainder of the paper is structured as follows. In Section 2, we give a brief overview of the process, Section 3 explains how the repository is generated and filled, while Section 4 describes its usage. A brief overview of related work is given in Section 5. The paper ends with a conclusion in Section 6. Throughout most of the paper, we have used a two-column style, where the left column contains explaining text, while the right one shows figures taken from the example scenario.

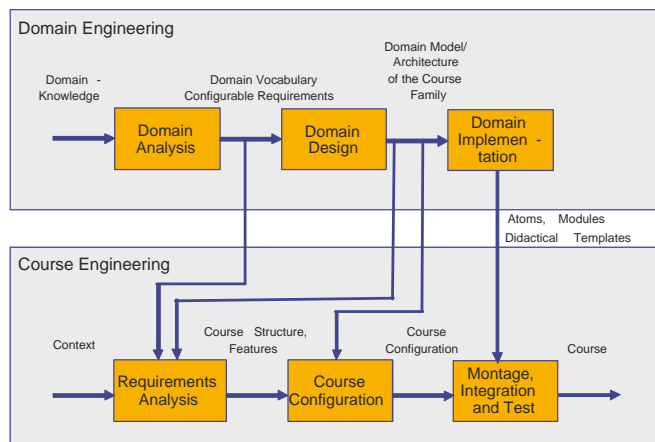


Figure 1. The process model

2 OVERVIEW OF THE PROCESS

Our approach proposes a development process [2] based on software engineering *for* and *with* reuse. It aims to support courseware reusability by giving an explicit support for both engineering for reuse and engineering with reuse. Figure 1 gives an overview of the SCORE development process for courseware. The process is divided into two main parts: the domain engineering process (engineering for reuse) and the course engineering process (engineering with reuse). The domain engineering process is conducted by the community and consists of the phases: domain analysis, design and implementation. In the domain engineering process instead of considering the development of one course we consider a domain or a family of courses. Thus, in this phase the community sets up an infrastructure or a framework for the cooperative development and exchange of reusable learning units. On the other hand, the course engineering process considers the development of one specific course for a specific context. It is based on the results of the domain engineering and consists of the phases requirements analysis, course configuration and montage as well as test and integration.

¹ This work was partially funded by the *ViKar*-Project [14] within the program *Virtual University* by the state of Baden-Württemberg.

² Institute for Program Structures and Data Organization, Universität Karlsruhe, D-76128 Karlsruhe, Germany, {kleinm, ateyeh, koenig, muelle}@ipd.uni-karlsruhe.de

3 DOMAIN ENGINEERING: CREATING AND FILLING THE REPOSITORY

Domain Analysis: Collecting Domain Terms. The first step when engineering the domain is domain scoping. As typical learning domains are very large, the members of the community try to delimit it by collecting terms that describe important parts of the content. In our example, they agree upon a set of terms concerning normalization as shown in Figure 2. Also, categories of material types they expect and want to support are collected. This could be, e.g., sentence, proof, example, definition, explanation, exercise, motivation, and solution. These two aspects, i.e., content and material type, are important when dividing existing material into reusable units.

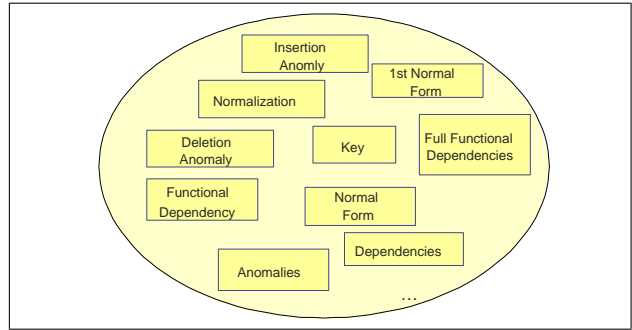


Figure 2. Collection of domain terms

Domain Design: Structuring the Domain. The goal of this step is to organize the brainstormed list of domain terms and material type categories from the previous section. To structure the domain knowledge into a domain ontology, the community inserts instances of the *isSuperTopicOf* relation between the domain terms to set up a hierarchical, tree-like structure of terms as depicted in Figure 3. In our domain for instance, anomaly is a super topic of deletion anomaly. In a second step, dependencies between terms are introduced. If a term t_1 is understandable only, if term t_2 has been understood, an instance of the relation *isPrerequisiteFor* has to be inserted between term t_2 and t_1 . Figure 3 also shows the pre-conditions in our domain. If someone wants to understand the 2nd normal form, he needs to have knowledge of the terms 1st Normal Form and Non-Key Attribute. As *isPrerequisiteFor* is a transitive relation, the remaining terms below dependencies need to be understood, too.

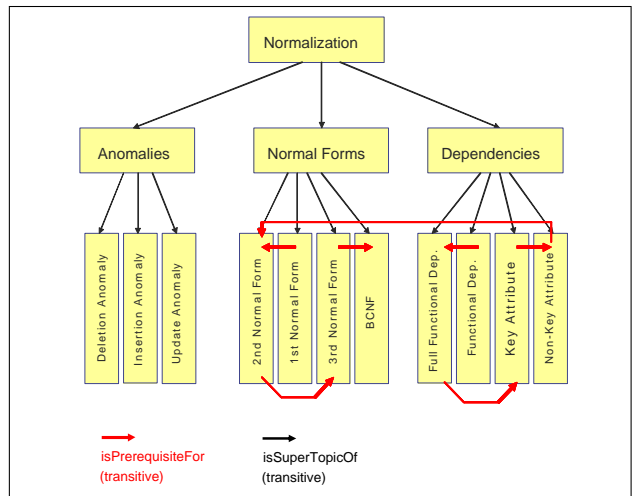


Figure 3. A domain ontology

Besides the content, the categories of material types are also structured in this step leading to didactical templates. In general, these define possible orders of material types to be used to set up a didactically sound course. Thus, there are many different templates for different institutions, learning groups, and learning forms, which additionally can be adapted for the concrete course. Figure 4 shows an example of such a didactical template for a rather practical course, for example a small-class tutorial: The course starts with an overview and a motivation for the topic. Then, each subtopic is first introduced theoretically with an explanation or a definition. Afterwards, it is practiced with an exercise or an example. Finally, a concluding example as well as an exercise with its solution are given.

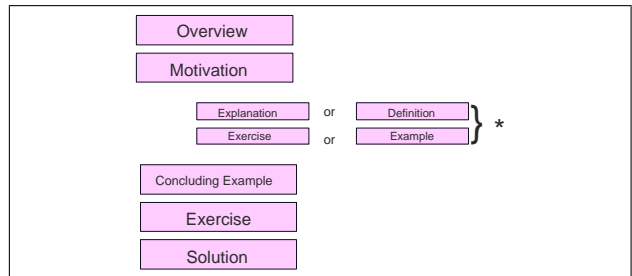


Figure 4. A didactical template

Domain Implementation: Assigning Material to the Terms. The last step of domain engineering is the implementation of the domain by assigning concrete learning material to the domain terms designed in the previous steps. Commonly, existing database courses today consist of large, monolithic blocks, which are often expensively created for one target group only. In most cases, these courses are sets of large Powerpoint files with hundreds of slides covering the content of a whole semester. To gain a repository of reusable components, it is inevitable to split up these courses into small, self-contained units. Generally, this is a very difficult process because the vague definition of "self-contained" does not offer

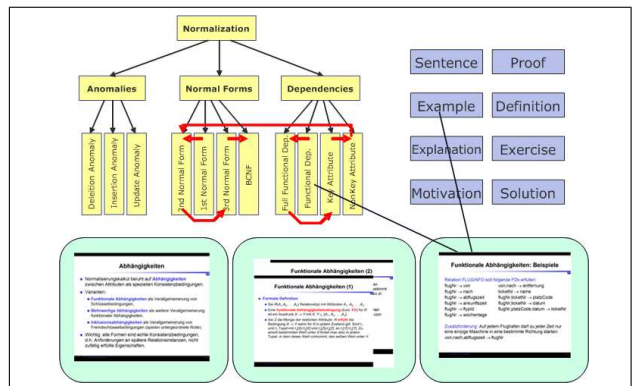


Figure 5. Learning atoms

any practical help to do the splitting. In contrast, in our approach, the two collections of domain and material type terms provide a concrete instruction for this process: Generally, a part is extracted as *learning atom*, if it belongs to exactly one content term and one or more material type terms. For our example, we decomposed among others an introductory database course held at the University of Karlsruhe, which is available as a set of seven Powerpoint files with approx. 100 slides each [8]. In Figure 5, three examples of the resulting atoms (depicted as rounded rectangles) are shown. The third one consisting of only one slide is assigned to the terms **Functional Dependencies** and **example**. Generally, the found atoms consisted of one to five Powerpoint slides and could be filed under one or at most two material types.

Resulting from the domain ontology and the assignment of atoms to domain terms, *learning modules* are defined implicitly and need not be composed manually: Each domain term t leads to a *learning module* containing all atoms and learning modules dealing with that term t . Figure 6 shows the module of the term **Functional Dependencies**, which consists of three learning atoms, only. Note that the third atom (the sentence about functional dependencies) stems from material of another institution, here from a database course of the AIFB [12]. An example of a recursive module can be seen in Figure 7: In the module of the more general term **Dependencies**, we can find the above-mentioned module on functional dependencies as well as modules of the other subtopics of **Dependencies** and atoms dealing with the content directly (for example a general explanation for dependencies).

It should be noted that domain implementation is not a one-time-only activity but an ongoing process. Initially, when the repository is being created, existing material is decomposed and stored. Over time, as new material is being created, this, too, is added to the repository.

4 COURSE ENGINEERING

Now that we have explained how the repository is built up, let us take a look at how it can be used to generate new courses.

Requirements Analysis. When an instructor plans to develop a new course, in a first step, she will decide on which topics to cover. Here, the domain ontology proves to be helpful. She browses through the ontology, deciding which modules to include in her course. For each module, available submodules are displayed, which she can choose to include or ignore in her course. Once all the topics are chosen, as a second step, she needs to define a structure for the course, i.e. she needs to impose an order on the modules she has chosen. Again, the system can help by notifying the user, if her order violates any of the *isPrerequisiteFor* relationships. Figure 8 shows the modules chosen for a course on normalization. All three submodules of normalization were selected, however, the instructor has decided to cover normal forms up to the third normal form only, thus, the submodule on BCNF was not selected. Note, for instance, that Chapter 1.3 on normal forms comes after Chapter 1.2 on dependencies as the 2nd normal form needs an understanding of non-key attributes, for example (compare prerequisites in Figure 3).

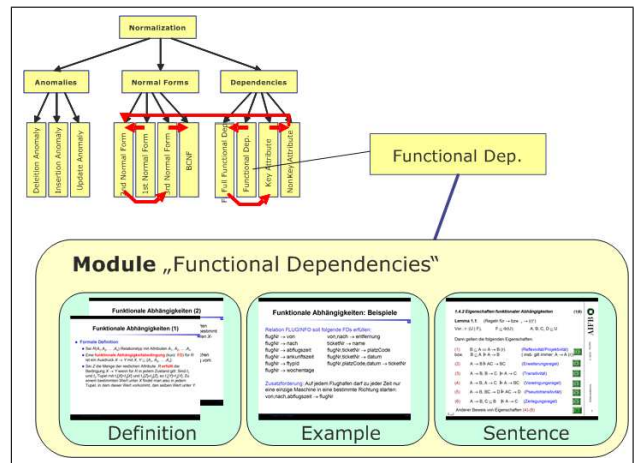


Figure 6. A learning module

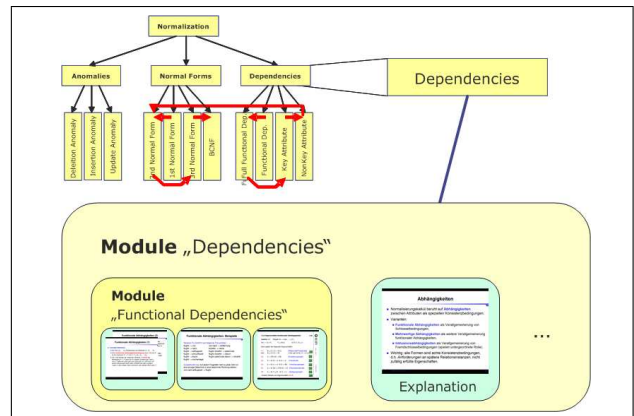


Figure 7. A recursive learning module

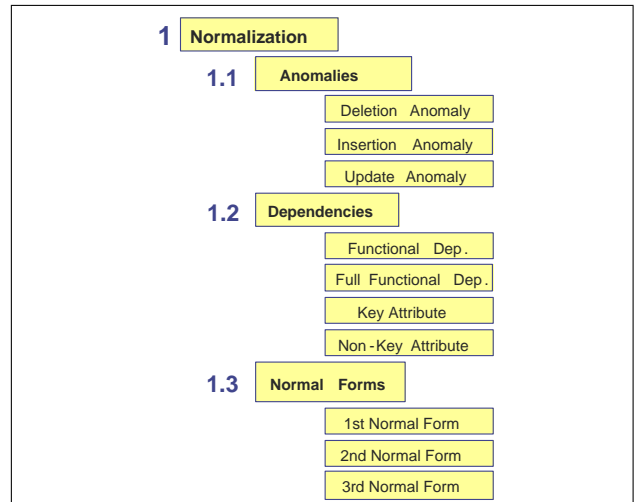


Figure 8. A course structure

Course Configuration. The inputs to the next step are on the one hand the course structure as described above and on the other hand a didactical template as depicted in Figure 4. These two are merged together semi-automatically as is shown in Figure 9. Now, this structure is filled with content. The system suggests suitable learning atoms for each of the (domain term, material type term) pairs. For our sample course, the instructor needs, among others, an example for the third normal form. As can be seen in Figure 10, three of the examples are stored in the repository and are now presented to the user. The instructor chooses one of these. Once this has been done for all parts of the course, the chosen atoms are automatically combined to form a course (see Figure 11). It might be necessary to add new materials to the repository, if the repository does not contain any learning atoms for a certain (domain term, material type term) pair or if the instructor does not like any of the atoms offered.

Course Montage. The result of the previous step is a semi-finished course. It is still necessary to add overview and summary atoms, to provide transitions between atoms, and to adapt notation or layout. Once this is done, finally, a complete course has been created.

5 RELATED WORK

The domains of software engineering, courseware engineering, and instructional design [6] are the main directions that influence our approach. We are applying domain engineering techniques [5] to the domain of courseware development. Most existing courseware development systems lay their emphasis on providing a systematic process to developing a new course (see [3] for an overview). In contrast, our approach aims at developing and exchanging **reusable** and **adaptable** courseware that can be used to develop courses in different contexts. The ARIADNE project [10, 7] supports the reuse of courseware with a repository of so-called learning objects, which are comparable to our learning atoms. However, our approach allows for a better structuring of the courseware resulting in a better guidance for the domain and course engineering process. Other projects like [9, 4] distinguish between modules and atoms. However, they offer less guidance than our approach on how to decompose learning material.

There exist also some initiatives (see [1], [13]) that are developing courseware portals for the domain of database systems, our application domain. The learning modules offered in those portals are coarser grained than most of our modules, though. This eases the direct usage of the modules by learners. However, it does not support the courseware engineering part as well as our system does.

6 SUMMARY AND CONCLUSION

In this paper, we explained how a repository of reusable learning objects for database courses can be created, filled, and used. Within our SCORE project, key components supporting the process have been implemented already. The remaining components are currently being developed and are expected to be available in early summer 2003.

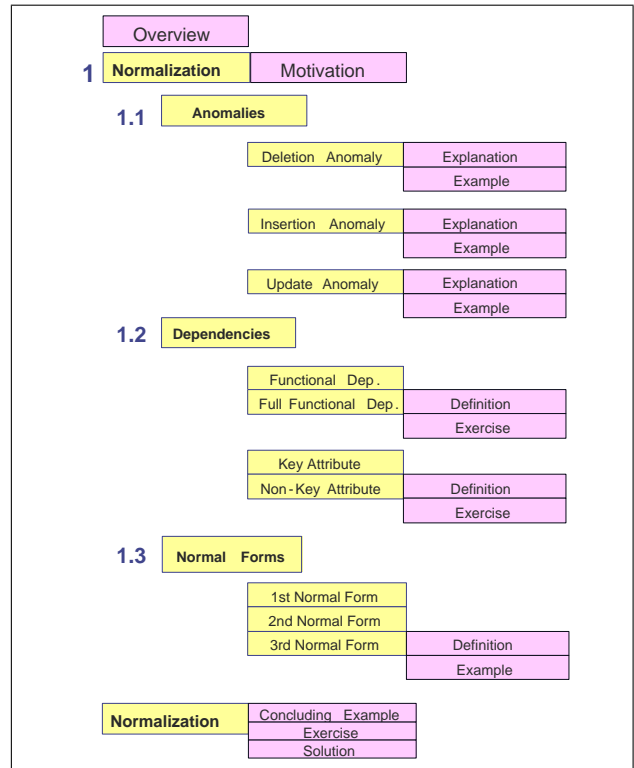


Figure 9. A complete course structure

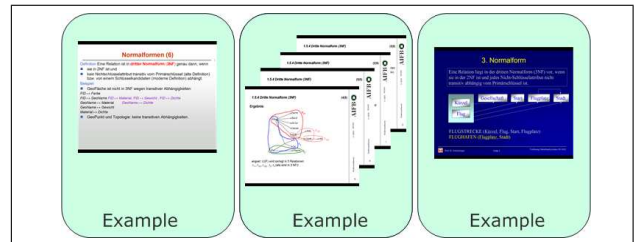


Figure 10. Example atoms

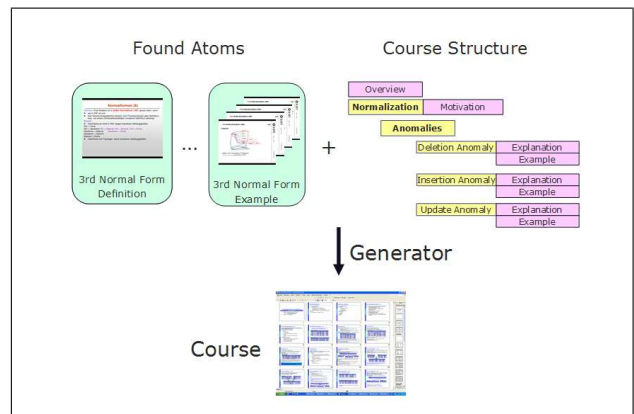


Figure 11. Course configuration

REFERENCES

- [1] ADIA Project: Virtual Course on Databases. Bildungsportal Sachsen, <http://www.imn.htwk.leipzig.de/~ed626/kudrass/elearning.html>
- [2] K. Ateyeh, M. Klein, B. König-Ries, J. Mülle: A Practical Strategy for the Modularization of Courseware. Accepted to: AELM - Workshop on Adaptive E-Learning and Metadata, 2nd Conf. Professionnelles Wissensmanagement, Luzern, 2-4 April 2003
- [3] S. Bostock: Courseware Engineering - an overview of the courseware development process. http://www.keele.ac.uk/depts/cs/Stephen_Bostock/docs/atceng.htm
- [4] The Candle EU-Project - Collaborative and Network Distributed Learning Environment. <http://www.candle.eu.org/>
- [5] Domain Engineering: A Model-Based Approach. Carnegie Mellon University, Software Engineering Institute, http://www.sei.cmu.edu/domain-engineering/domain_engineering.html
- [6] Definitions of Instructional Design. Adapted from: Training and Instructional Design, Applied Research Laboratory, Penn State University, 2002, <http://www.umich.edu/~ed626/define.html>
- [7] Erik Duval: An Open Infrastructure for Learning - the ARIADNE project. <http://www.cs.kuleuven.ac.be/cwis/departement/personeel/>
- [8] P.C. Lockemann: Introduction to database systems. Second part of the course on Telematics and Database Systems (in german: Kommunikation und Datenhaltung, Computer Science Department, Universität Karlsruhe, <http://www.ipd.uni-karlsruhe.de/Lockemann>
- [9] Müge Klein, D. Sommer, W. Stucky: Blended Learning in der Hochschule, GI-Jahrestagung 2002, pp. 270-274
- [10] D. Schwabe, C. Rossi: The Object-Oriented Hypermedia Design Model. Communications of the ACM, August 1995, Vol. 38, No. 8.
- [11] SCORE-Project: A System for Courseware Reuse. <http://www.ipd.uni-karlsruhe.de/SCORE>
- [12] W. Stucky: Introduction to database systems. First part of the course on Applied Informatics I (in german: Angewandte Informatik I), Institute of Applied Informatics and Formal Description Methods, Universität Karlsruhe, <http://www.aifb.uni-karlsruhe.de/>
- [13] Top University e-Learning International Program. <http://www.big.tuwien.ac.at/projects/tuelip.html>
- [14] ViKar: Virtueller Hochschulverbund Karlsruhe. <http://www.vikar.de>