Issue 2
2005

Software Engineering and eLearning: The MuSofT Project [1] www.musoft.org

Ernst-Erich Doberkat[a], Gregor Engels[b], Jan Hendrik Hausmann[b], Marc Lohmann[b], Jörg Pleumann[a], Jens Schröder[a]

[a] Dept. of Computer Science,
University of Dortmund,
Germany

ernst-erich.doberkat@uni-dortmund.de , joerg.pleumann@uni-dortmund.de , jens.schroeder@uni-dortmund.de

[b] Dept. of Computer Science,
University of Paderborn,
Germany

engels@uni-paderborn.de , hausmann@uni-paderborn.de , mlohmann@uni-paderborn.de

urn:nbn:de:0009-5-2016

Abstract

eLearning supports the education in certain disciplines. Here, we report about novel eLearning concepts, techniques, and tools to support education in Software Engineering, a subdiscipline of computer science. We call this "Software Engineering eLearning". On the other side, software support is a substantial prerequisite for eLearning in any discipline. Thus, Software Engineering techniques have to be applied to develop and maintain those software systems. We call this "eLearning Software Engineering". Both aspects have been investigated in a large joint, BMBF-funded research project, termed MuSofT (Multimedia in Software Engineering). The main results are summarized in this paper.

Keywords: Software Engineering, learning object repositories, eLearning community portals, reusability and interoperability, new media in technical education

1 Introduction

Software has become a key element in all aspects of day-to-day life. Thus, a high quality of software should be one of the most important objectives of any software development and maintenance activity. To achieve this, high standards in the education of people, who are responsible for and involved in developing software systems, are indispensable.

Software Engineering is the subdiscipline of computer science that deals with concepts, techniques and tools for supporting the development of high quality software systems. Software Engineering has gained an increasing importance during the last decade. In the meantime, nearly all computer science departments have established a chair in Software Engineering and changed their computer science curricula to contain several courses and practical trainings in Software Engineering.

Licence: fDPPL Any party may pass on this Work by electronic means and make it available for download under the terms and conditions of the free Digital Peer Publishing License. The text of the license may be accessed and retrieved at http://www.dipp.nrw.de/lizenzen/dppl/fdppl/f-DPPL_v1_de_11-2004.html.

However, teaching Software Engineering is a tough problem. This is due to the fact that benefits of Software Engineering concepts only become visible and understandable for students if they are applied to realistic problem scenarios of appropriate size and complexity. Those realistic scenarios are hard to present in classroom situations or even plenary lectures. On the other side, due to the huge number of students in the first years of computer science studies, it is not realistic to send all of them to industrial software projects.

This distance between realistic problem scenarios and didactical limitations can now be bridged by eLearning concepts and particularly by the use of multimedia techniques. This was one of the main objectives of the MuSofT (Multimedia in Software Engineering) project, the results of which are reported in this contribution. We will start with an overview of the objectives and organizational structure of the MuSofT project in Section 2.

Our eLearning concepts for teaching Software Engineering topics, which we termed *"Software Engineering eLearning"*, are presented in Section 3. We will report about three selected approaches within the MuSofT project to deploying multimedia and eLearning techniques for teaching Software Engineering concepts.

- First, we will illustrate how dedicated videos help to teach requirements elicitation on one hand and the usage of complex software development tools on the other hand.
- Second, we will illustrate how animations support the teaching of complex data structures and algorithms. Particularly, the dynamic behavior of data structures can intuitively be visualized by animations.
- Third, we show how specially designed lightweight modeling tools support an easy access to specific Software Engineering concepts and ease their understanding by students substantially.

eLearning means to use the computer and appropriate software systems to support students during their process of learning and understanding new concepts in a certain discipline. Thus, independent of a certain discipline, those eLearning supporting software systems have to be developed. This led in the past years to a huge number of software solutions ranging from complete eLearning management systems, over discipline-specific internet portals up to small dedicated support tools. Most of these tools, environments, and software solutions ignored the fact that all these software systems should be build according to well established Software Engineering principles and standards. Only this would have ensured that the developed software fulfills minimum quality standards like e.g. portability, interoperability, adaptability, or reusability.

We call this aspect *"eLearning Software Engineering"*. In Section 4, we will report about different techniques to ensure a high quality of eLearning support environments. This ranges from the development of dedicated tools over technical and legal issues to ensure interoperability of eLearning material up to the development of a comfortable internet portal for accessing eLearning material over the web.

The constructive process of creating eLearning material or dedicated software support tools always needs to be accompanied by an analytical evaluation process in order to check and guarantee that constructed solutions really improve the learning process of students. Thus, a continuous critical evaluation with a special emphasis on the gender aspect was also an important aspect within the MuSofT project and will be reported on in the following sections.

The paper is rounded up with a conclusion in Section 5 and some ideas for future exploitation of the MuSofT results.

2 The MuSofT project

The call for proposals in summer 2000 made it clear that the funding agency wanted to fund projects emphasizing the following two key aspects. On the one hand, the projects were to use multimedia for teaching the respective disciplines, covering as wide a part as possible of the subject. On the other hand, the projects had to guarantee a lasting contribution to academic teaching, even supporting other forms of teaching as in e.g. continuing education. All this was to happen given a potentially unstable financial background after funding would have expired, and given a somewhat dynamic technological development in the area of multimedia.

We decided that we wanted to focus the project thematically on a stable core in teaching Software Engineering, covering as much ground as we could in a reliable and attractive way. We decided furthermore that we wanted to jointly pursue the two goals of developing methods for teaching and laying the ground for a firm infrastructure on which to base further developments including broadening what we soon called the *customer base*. We discussed the addresses: would we want to talk to the students directly, or would we address our colleagues, addressing their needs and providing them with the material, which they could take and use it for enriching their own? Given the diversity of approaches to teaching Software Engineering, the multitude of programs into which concepts from software technology are integrated, and the potential goal of supporting continuing education with its very different assumptions on the participants' knowledge we decided that we did not want to address students directly but rather offer help to our colleagues. Thus, we ended up with the ambitious goal of providing a service to the German-speaking Software Engineering community.

A necessity for providing this service was a central, well-known distribution site on the Internet where authors could offer their material and potential users could easily find material suiting their requirements. We have termed this distribution site the *MuSofT portal*. Its goal is the management and distribution of individual learning objects contributed by the various partners and the facilitation of sustainable reuse of the material inside and outside the MuSofT community.

Although addressing teachers rather than students, we had to make some assumptions concerning the students' base knowledge that could be taken for granted. We assume that the students would have taken the introductory courses in computer science, so that they would be familiar (if not fluent) with one object-oriented language, in this way posing minimal demands on the students' linguistic competence. Students from other engineering disciplines usually meet this assumption as well, and since our subject is usually in one or the other form a topic in most engineering curricula, we decided to also include Software Engineering as a service to teaching engineers into our project. We did not stop here, either: teachers' education for teaching computer science in secondary schools should incorporate topics from Software Engineering, providing students with a glimpse at the realities of software construction mirrored through sound engineering principles. So we thought it a good idea to incorporate an explicit component dedicated to this class of students (and, since computer science is an emerging subject in German secondary schools, far from having the stability of, say, mathematics or physical education, we wanted to contribute to making some principles of software technology popular in secondary schools). Consequently, we wanted to address three sectors: Software Engineering as a subject in computer science curricula, in curricula for engineers, and finally as a topic in education.

Given the desire for as complete a coverage as possible, but constrained both by the budget and possibly by the number colleagues that could be incorporated into the project, we decided that we wanted to emphasize these topics part of which will be discussed in greater detail later on:

- the basic tasks in requirements analysis and realization (video based requirements elicitation, development of information systems, software technology in teachers' education)
- structures in software development (software architectures, design patterns, algorithms and data structures, software deconstruction)
- selected topics in the development process (the V model, software quality management, introduction to the Unified Process, project management)

The consortium consisted of well-known experts dealing with the following areas:

- Silke Seehusen (Fachhochschule Lübeck) provided models for design patterns
- Gunter Saake (Otto-von-Guericke-University Magdeburg) dealt with database issues
- Johannes Magenheim (University of Paderborn) was in charge of modules for educating computer science teachers
- Udo Kelter (University of Siegen) developed teaching material for the development process
- Andy Schürr (initially University of the Federal Armed Forces, Munich, then Technical University of Darmstadt) took care of data structures with their particular relevance for Software Engineering
- Fritz Schmidt (University of Stuttgart) focused on process models with emphasis on their use in the engineering discipline
- The authors were in charge of requirements engineering (Gregor Engels), and of tools for software architecture and for process modelling (Ernst-Erich Doberkat).

Given the list of topics, it is clear that we had to exclude a great many issues that we would have liked to address, but that we could not. Yet, this omission is not necessarily permanent, given that we plan a systematic extension that will be discussed briefly in Section 5.

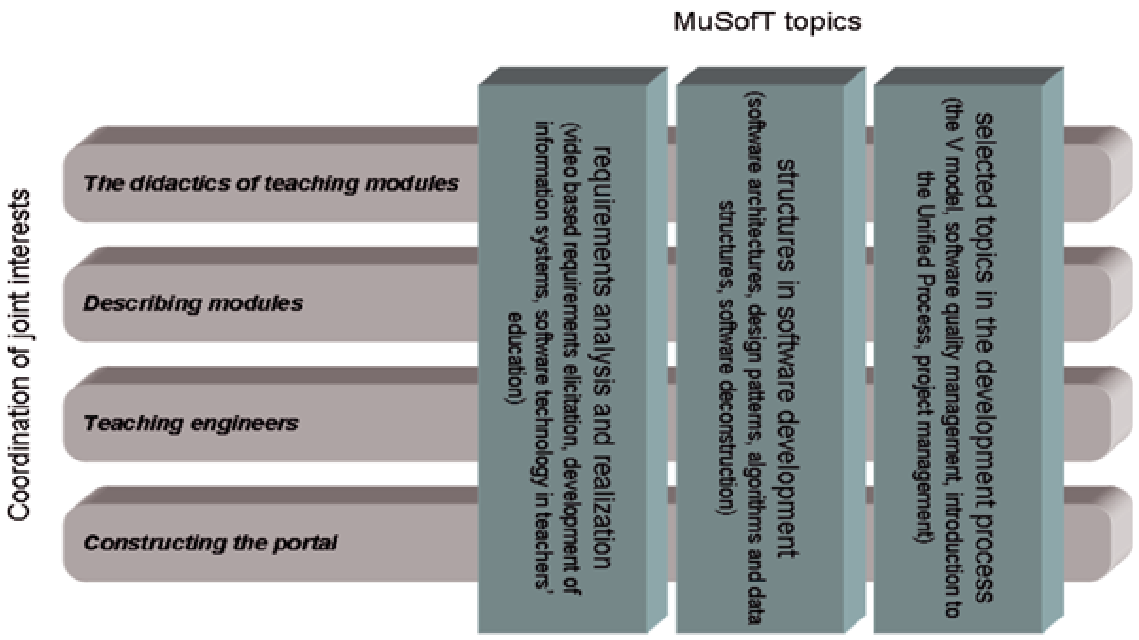

MuSofT topics
Coordination of joint interests
The didactics of teaching modules
Describing modules
Teaching engineers
Constructing the portal
requirements analysis and realization
(video based requirements elicitation, development of information systems, software technology in teachers' education)
structures in software development
(software architectures, design patterns, algorithms and data structures, software deconstruction)
selected topics in the development process
(the V model, software quality management, introduction to the Unified Process, project management)

Figure 1 - Structure of the MuSofT project

The topics handled by the MuSofT project may be visualized as a vertical structure in which teams located at the project partners' institutions would work. However we saw that the project might disintegrate if there would be only teams working in isolation. Then we would have results that would be superficially coordinated, possibly through a common vocabulary, but there would be no proper integration, rendering the whole enterprise a project unto its own, which in turn would jeopardize our intention to provide a service to the German speaking software technology community. This leads us to propose an additional horizontal structure with tasks that are of joint interest which were organized as committees (Figure 1).

Topics of concern included:

- **The didactics of teaching modules:** the main concern was that we would not find a common language and a common didactic approach, manifested in the concern that we would use a non-compatible terminology, and that we would violate fundamental didactic principles (after all, we had people from computer science education on our team). This committee was in charge of helping to establish a common didactic understanding.
- **Describing modules:** aiming at making our modules available to the public and having to discuss the development within the consortium we wanted to find a common way of describing the content of our teaching units. Scrutinizing different proposals from the literature we settled for the LOM approach. This entailed then finding a suitable subset of the attributes and a viable way of performing the attribution, it will be described in Section 4.2 in greater detail.
- **Teaching engineers:** the incorporation of engineers was planned essentially through two measures. They should be exposed to a basic level of techniques, taking into account that engineers want to apply these techniques in the first place, and they should exercise the material through specially selected exercises or projects. This

committee was in charge of guiding the consortium through this process. It turned out, however, that the basic level of exposition was of interest to computer scientists as well, and that the specially selected exercises and projects could be taken from the supply of computer science. So we abandoned this group rather early in the project.

- **Constructing the portal:** we wanted to focus on collecting our material and offering an entry point for exporting it; we will describe this key point in Section 4.4.

3 Software Engineering eLearning

In this section, we will report about different approaches within the MuSofT project to deploy video and animation techniques as well as dedicated lightweight tools to support teaching of Software Engineering topics.

3.1 The use of video techniques

Requirements specification a branch of Software Engineering that only make sense if you consider reality. In an ideal world, a customer would provide a complete set of technical, functional, and non-functional requirements to a software developer (or a software company). The developer could then proceed to create a solution based on this input. In the real world, however, requirements specification techniques is a field of research in its own right because the elicitation, notation and analysis of requirements is an intricate process which is prone to problems like information overflow, contradictions, misunderstandings, and unjustified assumptions. In fact, the CHAOS report of the Standish Group [Standish 1995] lists problems in the requirements gathering phase as the top reason for project failures. It is thus important to teach students the use of techniques for requirements specification.

In teaching requirements engineering, we can identify an inherent obstacle. Most of the problems in gathering the requirements stem from the fact that the requirements are formulated and posed by persons that are unfamiliar with the process of requirements capture, have no knowledge of the technical terminology used in Software Engineering, and cannot distinguish requirements towards a software system from the working context surrounding it. There is furthermore the problem that the input is typically incomplete and can even be contradictory if multiple stakeholders have been interviewed.

A good lecturer, however, is always trying to present consistent, precise and complete information to his students. He is aware of the technological terms and automatically categorizes information according to his knowledge. If such a lecturer tries to present a requirements engineering problem, he will find it very hard - if not impossible - to keep this knowledge from influencing his presentation in a way as to give students hints on the solution of the problem. In other words, it is impossible for him to play ignorant.

The use of video techniques is a way of solving this dilemma. By relying on an external medium for the presentation of the problem domain, the lecturer can be sure that no solution hints slip into the presentation. This effect is maximized if videos can be obtained from external sources that are not influenced by the intentions of requirements engineering. Thus, the influence of the lecturer is eliminated from both the production and the presentation of the example. A video does furthermore present the full complexity of realistic work surroundings and the activities carried out there. To handle this audiovisual complexity, to avoid information overload, and to form the right abstractions is an important part of requirements engineering. If the problem domain is presented by a text instead of a video, this aspect gets lost since the text usually already comprises abstractions from the complex reality of the problem.

In the MuSofT project we have heavily made use of video techniques to present problem domains for requirements engineering techniques. Different levels of complexity have been chosen to supply material for different learning situations. For the lecture, we could obtain a video from a machine manufacturer (originally intended for marketing purposes) which describes the operations of an automated storage system for a hospital. We have used this video to present the domain and apply the techniques of requirements specification to build UML models of the relevant static structures and operation flows of the domain. Students were then requested to apply these techniques in exercises. For these exercises, we employed videos showing playing situations from a board game (Mississippi Queen, [Hodel 1997]). Again, the videos formed the only description for this domain and students had to create models based on these videos. Another video presented to students shows an interview in which information about the system to be developed is given by a customer to a software developer. In a final example, we wanted to present more specialized problems for the application of goal-oriented requirements capture techniques. Since these are advanced concepts, we had to create videos showing scenes from a business context but with problems for which the application of goal-oriented techniques is especially profitable. These videos serve both as a motivation for goal-oriented analysis and as a basis for its application.

Our experiences with employing these videos in teaching were very positive. We have provided this course to almost 2000 students over the last four years. All technical problems with presenting or distributing the videos were easily overcome. Direct feedback by the students (as recorded by questionnaires) revealed the videos to be a very welcome variety to the usual lecture format. However, the students also realized that the exercises based on the videos present additional problems. They had to cope with higher abstraction distances, incomplete information, and overall with the fact that there is no single optimal solution. The students were uncomfortable with these problems: up to a 33% requested "more technical examples", up to 55% indicated problems in "debating different solutions". These numbers indicate that we were able to sensitize the students for the problems of requirement specification. This sensitivity forms a very good motivation to learn Software Engineering techniques.

Another example for the representation of reality in teaching Software Engineering techniques is the usage of tools. Knowledge about and skills in some standard tools are a standard goal of Software Engineering education. Only the combination of knowledge on why and when to do something and the capability to actually do it make a complete Software Engineer.

While many high-level concepts of Software Engineering can (and often have to) be described in an abstract way, the use of tools is necessarily a very practical and realistic kind of teaching. It is on the other hand also a kind of teaching which is below the usual abstraction level of university teaching. Thus, many lecturers are bored by these details and skip them in favor of more interesting topics.

Video techniques once again offer a solution for this specific problem. By producing a suitable teaching video which contains instructions for the use of a tool, the students have a very practical help in applying the tools for their exercises and the lecturer is unburdened from teaching tool details again and again.

In the MuSofT project, we developed a multi-part video presenting the concept and practical use of the CVS version control tool [Kelter 2002]. Version control is a crucial aspect of virtually every software development project and CVS is the most prominent tool to support it. Thus, using CVS can be considered a basic skill for all software developers. By combining animations which illustrate the fundamental concepts of version control and actual screen movies providing detailed instructions on how to carry out certain tasks or react to certain messages, the viewers get familiar with the system and the underlying concepts very quickly. Evaluations have shown these videos to be very popular with the students, especially as they perceive a smaller cognitive distance in comparison to explanatory texts and feel that these videos form a very efficient way of being accustomed to the program and its usage. The success of these videos is also reflected in their distribution: Several other universities have actively used these videos as part of their Software Engineering teaching.

3.2 The use of animations

Algorithms and data structures are fundamental concepts in Computer Science, and their understanding is a crucial aspect of Software Engineering education. Nontrivial algorithms typically include complex rearrangements of the data structure they operate on. A good example is the execution of insertion or deletion operations on an AVL tree: Inserting elements into or deleting elements from the tree not only affects a single node, but also triggers various rearrangements on other parts of the tree to ensure its balancing property. Thus, to impart knowledge about the AVL tree, its dynamic behavior must be understood, i.e. students need to know which rearrangement to execute for a given state and what the underlying tree structure looks like afterwards. The same holds for many other algorithms and data structures.

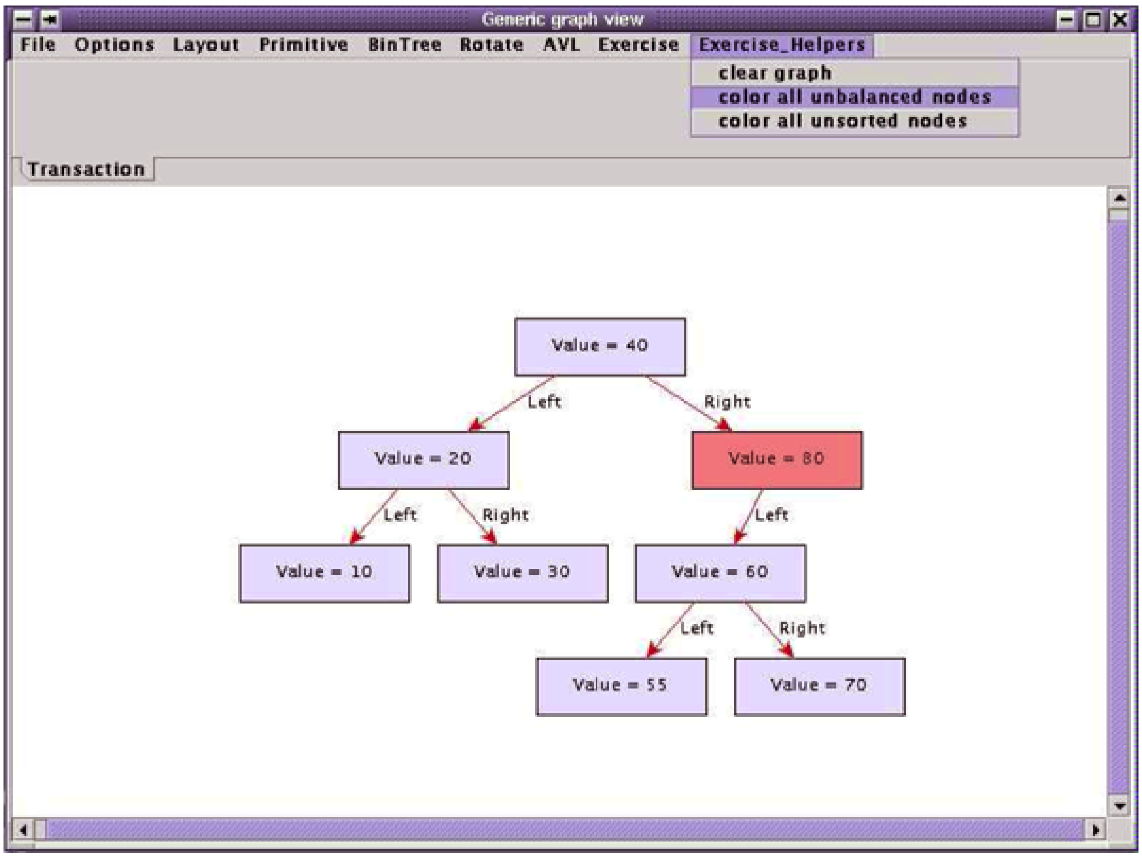


Figure 2 - Visual Interactive Data structure Environment, AVL tree

In the MuSofT project, we have employed interactive animations for teaching algorithms and data structures [Aschenbrenner 2003a]. We developed a visualization environment called VIDEA (Visual Interactive Data structure Environment for Animations, [Aschenbrenner 2003b]) to create interactive animations. We use the animations in different scenarios.

- Firstly, the animations are used to introduce certain algorithms and data structures during lectures or assignments. As an example, the aforementioned AVL tree is constructed in an interactive manner, starting from an empty tree. The lecturer or a student determines which nodes to add. The algorithm for balancing the tree is automatically performed, with all intermediate steps being visualized by the animation.
- Secondly, we created animations for interactive assignments: In this scenario, the students themselves have to perform the algorithm by manually manipulating the data structure. In the case of the AVL tree, the students have to balance a given unbalanced tree by performing the corresponding rotations on the tree. The animation immediately visualizes the results and provides feedback on correctness by highlighting those parts of the tree that hurt the balancing property (Figure 2).
- In a third learning scenario we have used animations during a lab course for visually debugging algorithm implementations the students had to develop in Java. The students worked in several teams, all of which had to solve the same problem: brewing as much beer as possible in a given time. For solving the brewery problem, the students had to consider several classical computer science problems such as the

knapsack problem or finding minimal spanning trees in graphs. The algorithms for the subproblems were combined into an algorithm for the overall solution and implemented in Java. A small Java framework connected each implementation to a set of Flash animations visualizing the current state. Figure 3 shows a screenshot of a sample animation.

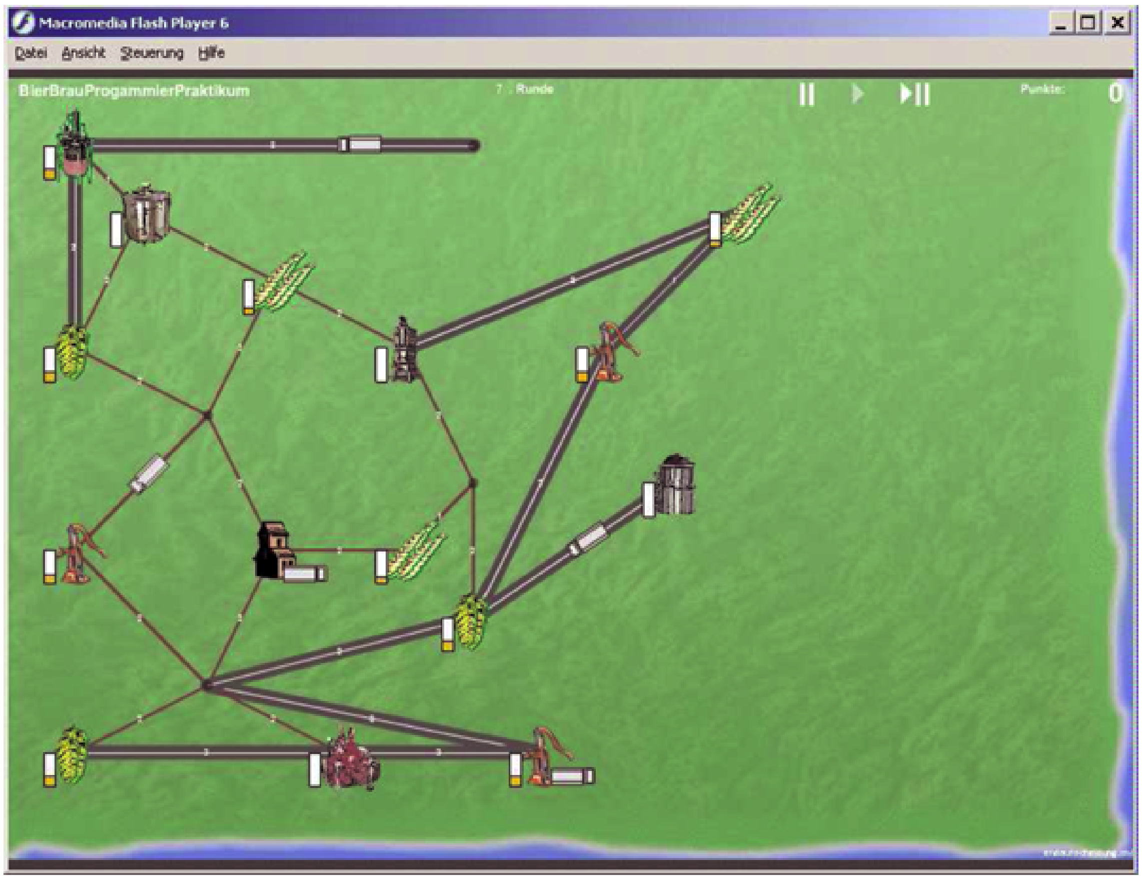


Figure 3 - Brewing beer - the knapsack problem

We have evaluated the application of animations for all three learning scenarios. The results of the evaluations are promising, since they show that the students are generally more motivated, and, in comparison to a control group doing assignments without animations, have a better understanding of the introduced algorithms and data structures. For an example algorithm (finding a shortest path in a graph) and an example data structure (AVL trees) the number of students who were able to correctly answer questions regarding the dynamic behavior was significantly higher in the group doing animated assignments than in the control group.

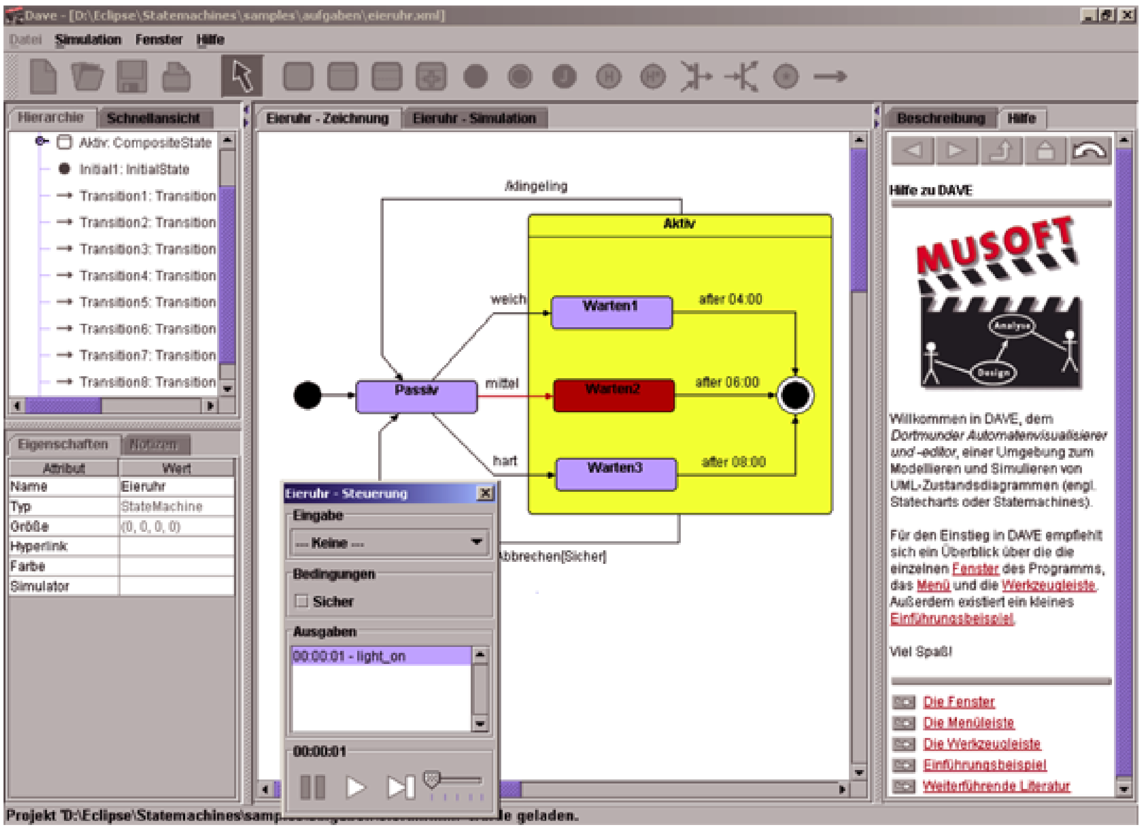
3.3 The use of dedicated tools

With today's software systems becoming more and more complex, teams getting larger, and the development itself being distributed in space and time, the importance of a good *model* of the system under construction is growing. The model serves as an architectural blueprint and as a communication means for all stakeholders participating in a project. It often decides on key factors of the resulting system, like correctness, reliability, security, or maintainability. Recent initiatives like OMG's Model Driven Architecture (MDA) [Frankel 2003] even make the model the central artifact of the overall development process. For students to be prepared for these requirements, it is necessary to teach them basic modeling concepts as well as concrete languages, the Unified Modeling Language (UML) [OMG 2003] surely being one - but not the only one - of these. If the size of models used, for example, during assignments approaches that of real-life problems, pen-and-paper work is no longer feasible. Tool support becomes necessary.

Unfortunately, the modeling tools typically used in industry, such as IBM's Rational Rose or Borland's Together, have significant drawbacks when applied in an educational setting. These drawbacks stem from the fact that professional modeling tools are rather *heavyweight* pieces of software, both in terms of their feature set and the hardware required to run them smoothly. Using such tools, there is a risk that merely the tool handling is taught instead of the particular language or method supported. If different tools are used for different notations, this situation becomes even worse, since the students have to be familiar with all these programs before being able to work effectively. The tools' user interfaces are quite complex, with lots of features aimed at efficient development in industry. Too many features will distract an audience when using the tool for demonstration purposes, for example during a lecture. In addition, for educational purposes it might sometimes be desirable to use a simplified notation. This would require the tools to be easily adaptable, which is usually not the case. Last but not least, commercial tools are expensive, which might pose an additional problem if several hundred licenses are required for a large computer science department.

These deficiencies of industrial tools have already been acknowledged for the field of programming environments. BlueJ [Kölling 2003], jGrasp [Hendrix 2004] and Dr. Java [Allen 2002] are examples of popular programming environments that were developed with introductory courses in object-oriented programming in mind. All three tools share the philosophy of a reduced feature-set and a simplified user interface, and could thus be considered *lightweight* when compared to the industrial ones. All three add certain features motivated by didactical problems usually encountered during introductory Java courses. BlueJ employs a UML-like interactive graphical visualization of Java classes and objects to foster an understanding of the semantics of these fundamental concepts. jGrasp adds to this idea a means of visualizing the state of data structures at run-time. Dr. Java is centered around a Java interpreter that can execute single Java commands without the need for writing a complete program and its notorious main() method.

For the field of graphical modeling, similar tools are rare. Fujaba [Nickel 2000] is an example of a UML CASE tool that provides a certain lightweightness when compared to Rational Rose or Together. Yet, it doesn't add many features to take care of didactical problems such as teaching the complex syntax and semantics inherent in a number of UML-based languages.



Willkommen in DAVE, dem *Dortmunder Automatenvisualisierer und -editor*, einer Umgebung zum Modellieren und Simulieren von UML-Zustandsdiagrammen (engl. Statecharts oder Statemachines).

Für den Einstieg in DAVE empfiehlt sich ein Überblick über die die einzelnen Fenster des Programms, das Menü und die Werkzeugleiste. Außerdem existiert ein kleines Einführungsbeispiel.

Viel Spaß!

Die Fenster
Die Menüleiste
Die Werkzeugleiste
Einführungsbeispiel
Weiterführende Literatur

Figure 4 – Screenshot of DAVE (Dortmund Automaton Visualizer and Editor)

This led the MuSofT project to the idea of developing a family of lightweight modeling tools targeted at educational use only. This family of tools, which has been realized at the University of Dortmund, currently encompasses structural and behavior diagram types of the UML, as well as process modeling and conducting based on the Unified Process [Jacobson 1998]:

- The Dortmund Automaton Visualizer and Editor (DAVE) [Pleumann 2004], is a graphical environment for working with Statecharts [Harel 1987], an extended version of the classical Moore and Mealy automata. A screenshot of this tool is shown in Figure 4.
- The Software Architecture Modeler (SAM) supports the graphical specification of structural and behavioral aspects of component-based software architectures. The notation used is a subset of the notation propagated by the UML 2.0 for that purpose.
- The Process Modeler and the Process Tutor [Kopka 2004] are tools for teaching the Unified Process. The former supports a student in tailoring the generic Unified Process to the needs of a particular project, going down to the level of roles, activities, and the artifacts produced by them. The latter is an easy-to-use process management tool that allows to conduct the modeled process in reality, for example during a lab course.

In addition to a restricted core feature set borrowed from existing modeling tools and particular attention being paid to usability concerns and a common user interface for all tools, several didactically motivated features have been incorporated. These features support the students in being acquainted with the particular modeling approaches. The

DAVE tool, for example, includes a sophisticated simulation engine that supports the *debugging* of a Statechart in a visual manner. The simulation provides insights into the complex execution semantics of the formalism: Active States and firing transitions are highlighted properly while the students feed input events into the system. Output events are recorded, and the whole simulation history is logged, so a simulation run can also be analyzed *post mortem*.

In order to support those students who have problems with the very high level of abstraction inherent in Statecharts and thus might not see their possible practical applications, the simulation can be supported by a multimedia representation of a real-life device. The embedded computer system of this device is supposed to be *controlled* by the Statechart. Several such multimedia devices already exist: A washing machine, a coffee maker, a glass-recycling machine, and a cash dispenser. They all give a vivid representation of the relatively abstract Statechart model as well as some feedback on the correctness of the model. In the case of the washing machine (Figure 5), if the laundry comes out clean, the student's model is likely to be correct - if the machine floods the room there is obviously something wrong with the part of the model that controls the valves.

Evaluations have shown that the lightweight MuSofT tools are particularly well-suited for assignments during a Software Engineering class or lab course. The evaluations were supported and surveyed by the Center for Higher Education Research and Faculty Development at the University of Dortmund [Kamphans 2004a, Kamphans 2004b] with a special emphasis on taking care of the gender aspect.

DAVE, for instance, was first used in the summer of 2003 during an Software Engineering class in which a number of visual formalisms were taught, Statecharts being one of them. About 100 students used the tool to carry out their assignments over a period of two weeks. One of the assignments incorporated the aforementioned washing machine. Following the assignments, a formal questionnaire was handed out to the students. The feedback gained from this questionnaire was very positive. The students appreciated the tool's simple user interface (89%) and the integrated simulation engine (90%). Most of them found that the simulation had improved their understanding of Statechart semantics (81%). The students would recommend the tool to colleagues (75%) and requested similar tools for other parts of the class (86%).

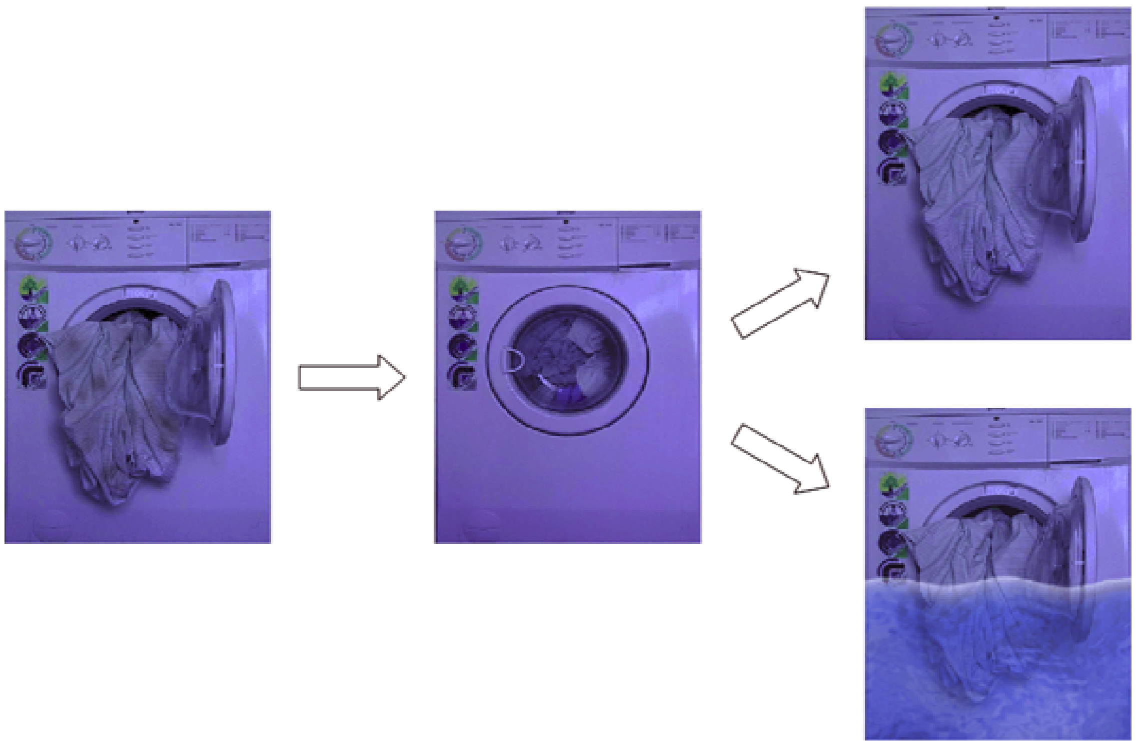


Figure 5 - Animations used for washing machine example

4 eLearning Software Engineering

Software Engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers. On the one hand, eLearning means are supported by different software systems and thus Software Engineering techniques have to be applied to develop high-quality tools supporting eLearning. On the other hand, Software Engineering techniques can be helpful for developing learning objects (digital entities that may be used for learning, education or training). Especially, if we have multimedia learning objects, which can be considered as a form of software, we can apply software development processes that allow us to develop learning objects in a structured way to achieve different qualities. As an example, we show in section 4.1 a learning object, which is an autonomous software system.

In the rest of this section, we want to consider the quality aspect of reuse in more detail. This aspect can be regarded from different perspectives. In Section 4.2, we explain how we enable reusability of learning objects within MuSofT by using existing standards. Further, we can view the problem of reusability from a legal perspective. To handle the legal aspect, we have developed a MuSofT license (section 4.3) that we use to publish our material. Our MuSofT portal for managing and distributing learning objects assists in all three aspects (section 4.4).

4.1 The development of dedicated tools

As mentioned before, the MuSofT project has developed a number of graphical modeling tools dedicated to Software Engineering education. Section 3.3 has given a brief overview on some of these tools from a user's perspective: In addition to the core modeling functionality, the tools were to include features that support the understanding of the modeling formalism itself. The simulation engine sketched for the DAVE tool is a good example; hypertext facilities included in all the tools are another. Yet, apart from fulfilling these rather functional requirements, some non-functional requirements had to be taken into account, too:

- Usability. The modeling tools had to be easy to use. The user interface complexity had to be significantly lower than that of professional tools. In the ideal case, all tools were to share a common user interface.
- Maintainability. Development and maintenance had to require as little effort as possible. The maintenance was of particular importance, since the tools still had to be maintained and improved when the project's funding had already ended.

To avoid implementation of each of the mentioned modeling applications individually and from scratch (thus repeatedly reinventing the wheel) and to also establish a common look and feel, a Java/Swing-based framework for lightweight modeling tools was developed. The framework provides two main components: A more-or-less fixed user interface including a working application frame, and a variable graphical language described by means of a *metamodel* underneath.

To be suitable for educational purposes, the user interface was designed to be simple, intuitive to use and non-distracting. The model itself occupies the largest part of the screen, because this is the application's focus. All graphical aspects of modeling, like moving or resizing elements or drawing graphical connections between them, are handled here. Everyday functions, such as loading, saving, printing, or adding new elements to the model, are easily accessible from a toolbar atop the diagram area, with clear graphical icons describing each function (Figure 4).

In addition to the main diagram area and the toolbar, the application provides a number of user interface components with more specific purposes:

- The *Navigator* shows a tree-like representation of the model (based on the nesting of elements) as well as an overview in the form of a map. It is useful for models the size of which is beyond a single screen.
- The *Inspector* displays the properties of the currently selected model element and allows their modification. It also supports the annotation of individual model elements by means of hypertext. Thus, the students can make notes on their model.

These two components are placed left to the diagram area, honoring the common principle of placing navigational items on the screen's left side. Both can be hidden when only the diagram itself is of interest, for example during a lecture. To the right of the diagram, a hypertext window can be shown. It allows annotating each model element with a hypertext page that is displayed when the element is selected. Possible applications are complex, annotated models which the students can study. It can also be used for handing out exercises to the students in digital form. Last but not least, it contains the online help for each tool.

The user interface provides different views on a model whose syntax and semantics is determined by an application-specific metamodel. This metamodel - together with the figures that make up its graphical representation - is easily *plugged* into the application frame by implementing each metamodel entity as a small Java class, defining its properties and the syntax rules it obeys to. Core Java classes for deriving application-specific metamodels from are already provided by the framework. These could be considered a meta-metamodel shared by all tools, although they are not on the meta-metamodel (M3) level in a strict technical sense. Thus, we usually refer to them as *basic metamodel elements*.

In order to have the same user interface work on all application-specific metamodels without adjustments, these basic metamodel elements provide introspection mechanisms that allow querying an element's properties and relationships with other elements, and possibly change them. These mechanisms are used for loading, saving, and printing models on a generic basis as well as for providing a foundation on which the aforementioned Navigator, Inspector, and Hypertext components could be built.

This framework architecture has already proven to be valuable. In addition to the MuSofT tool family mentioned in Sec. 3.3, several other modeling tools have been implemented over time, most of them by students. One example is the Petri Net editor PETRA. Originally started as a diploma thesis, it has grown into a full-featured simulator and analyzer since [Fronk 2004].

4.2 eLearning Standards

One important aspect Software Engineering is concerned with is the reuse of existing systems for building new software systems. A fundamental basis for facilitating sharing and exchanging software systems is a well-structured standardized description. Often the structures of these descriptions have to be adapted for different problem domains.

The aim of the MuSofT project was to develop a set of learning objects for Software Engineering courses that can be grouped according to the requirements of a special course at a university. Therefore, we needed a common conceptual data schema for the description of our learning objects with metadata to ensure a high degree of semantic interoperability of our descriptions. Further, our metadata descriptions should facilitate search, evaluation, acquisition, and use of learning objects by other instructors not concerned with the MuSofT-project. Instead of developing a proprietary conceptual data schema, we searched for existing eLearning standards for describing learning objects that we can customize according to our requirements. We decided to adapt the IEEE Learning Object Metadata (LOM) [IEEE 2002] standard. LOM is a broadly accepted standard that is part of the most important eLearning standards like the standards from the IMS Global Learning Consortium [2] and ADL SCORM (Sharable Content Object Reference Model) [ADL 2004].

In a next step, we specified the relevant characteristics for describing our learning objects and adopted the LOM standard accordingly. In order to maximize the semantic interoperability of our metadata descriptions, we carefully mapped our needed metadata information to the data elements of the LOM standard. Mainly we used the feature of LOM to specify vocabularies to restrict the content of some data elements. In these cases, we predominantly used existing, standardized vocabularies. E.g., LOM specifies data elements for classifying a learning object according to a classification system. In order to maximize semantic interoperability with computer science learning objects from other projects, we use the ACM Computing Classification System [ACM 2005] instead of a proprietary classification system. The ACM classification system is a broadly-accepted taxonomy that structures the field of computer science over different levels. As a result, we developed a conceptual data schema for MuSofT that is strictly conforming to the LOM standard, which means it consists solely of LOM data elements.

The LOM standard is intended to support consistent definition of metadata elements across multiple implementations, but does not include information on how to represent metadata in a machine-readable format, necessary for exchanging metadata. We use a representation of the metadata in XML (Extensible Markup Language) developed by IMS. The IMS representation is used by different eLearning applications and it is a part of the SCORM standard, which integrates different eLearning standards and specifications into one bigger, general standard. We can use the IMS standard without any changes except that we do not use all of the optional elements.

Beyond a data format for our conceptual data model, we need a standardized way to exchange digital learning objects including their metadata descriptions between different systems or tools. Two important content packaging formats are available today. On the one side, we have the IMS content packaging standard [IMS 2003] and on the other side the content packaging part of the SCORM standard. IMS content packaging is a part of the SCORM standard. The SCORM standard additionally allows exchanging information about the intended behavior of a collection of learning resources. We decided to use only the IMS content packaging format, because it is sufficient for us and, in contrast to the SCORM content packaging, it is implemented by different tool vendors (e.g. Blackboard, Centra, CourseKeeper).

4.3 Open content development

When making our work available to the public, we want our material to be freely usable and adaptable for educational purposes, but we do not want it to be exploited commercially without the authors' prior consent. Thus we are in a situation similar to the open source community, but the legal space in which we have to move is a little different: First, we are dealing with documents the character of which is somewhat extended compared to the traditional one. Second, we want our collection of teaching material to be modifyable and extensible by others, so that content may in the long run also be provided that was not produced exclusively in the MuSofT project. Third, we aim mainly at the German educational system, so that German law applies (true, German law undergoes some changes with respect to copyright issues, but its fundamental principles will not change). Given these observations, we asked the Universitätsverbund Multimedia (UVM), an agency run by the state of Northrhine-Westphalia that has been set up with the broad goal of furthering the use of multimedia in the state's universities. The request was to construct an open content license for our purposes. Alas - it turned out that we were breaking new grounds here - such a license had not been constructed before. But UVM complied with our request, co-operating with some specialists in that very new and apparently very exciting area of German law.

The present and the future content provided by the MuSofT project is protected under this new *MuSofT Open Content License* [Jaeger 2003]: use and modification are free for nonprofit educational purposes, just as we intended. The license is sticky, similar to the GNU General Public License (GPL) used in many open source software projects, since it propagates through modifications. It would also have been possible to use a Creative Commons license. Yet, Creative Commons has a strong focus on document-like content (text, audio, video, ...), while a significant portion of the MuSofT material consists of software. Thus, we needed a license that makes a distinction between source code and executable code and regulates the use of both. Finally, Creative Commons licenses have only been adapted to German law in mid-2004, where we needed to make provisions for proper licensing of our material much earlier than that.

We think that with the MuSofT license we have found a legal protection that does not constrain the future use of our material through either ourselves or our contributing colleagues.

4.4 eLearning Portals

Annotation of e-learning material by means of metadata and an appropriate licensing scheme were two necessary requirements for making sure that the material can undergo sustainable (re-) use, but they were not sufficient. A central, well-known distribution site on the Internet was required where authors could store their annotated material and potential users could easily find material suiting their requirements. We have termed this distribution site the *MuSofT portal*. Its goal is the management and distribution of the individual learning objects contributed by the various partners and the facilitation of sustainable reuse of the material inside and outside the MuSofT community. The MuSofT portal is not a learning management system, since the development of such infrastructure was not a concern of the project, and several learning management systems are already in use at universities.

Envisioned users of the MuSofT portal are teachers at universities or other institutions of higher education who are looking for multimedia Software Engineering material or want to distribute their own material. It was clear from the beginning that the initial users would recruit from the MuSofT consortium itself, but the later integration of external users was desired. We thus chose to distinguish among three different user classes:

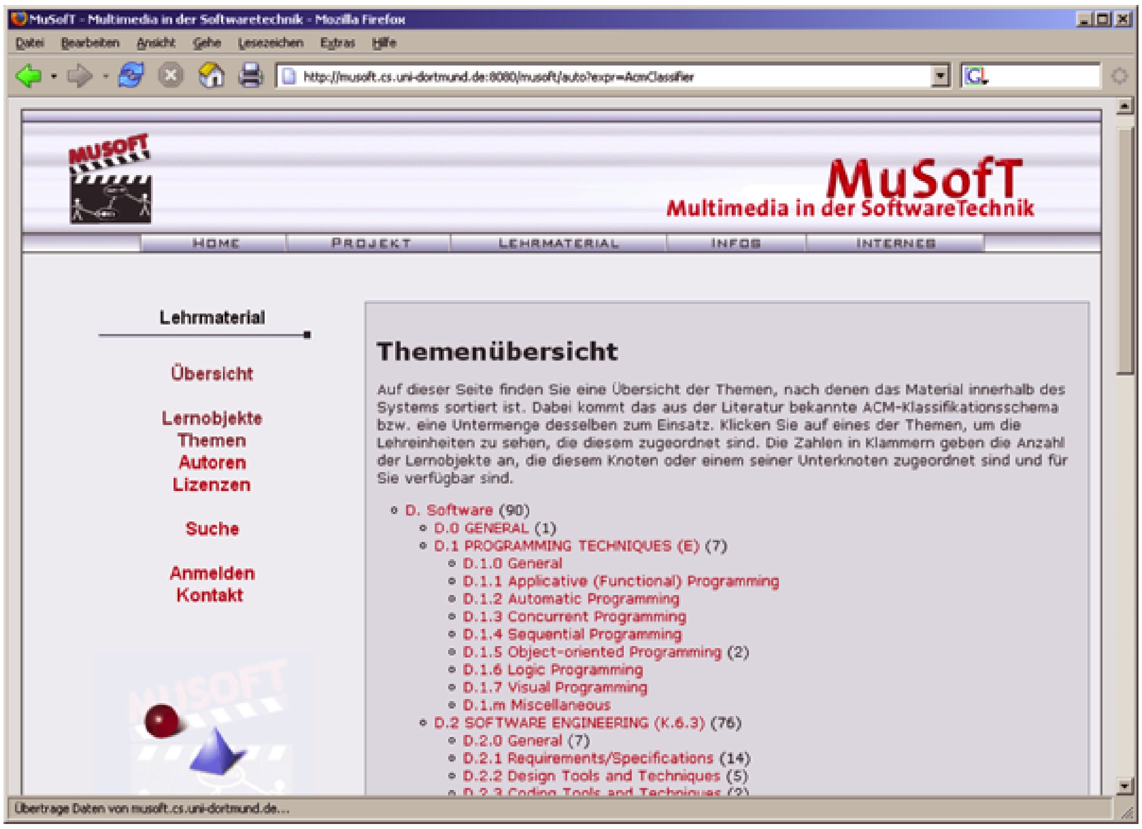
- *Consumers* search and download material for application in their classes. They can browse the contents of the portal by their ACM classification (Figure 6), by authors, or by granularity of material (Figure 7). Consumers can also perform queries for material based on any of the supported metadata fields. When the desired material has been found, its metadata can be inspected (Figure 8) and the material can be downloaded in the form of an IMS content package (Section 4.2) that includes both the learning object(s) and the corresponding metadata and licensing information. The standardized export format provides for easily importing the material into an existing learning management system. Consumers can also send feedback to the author of a learning object and request e-mail notifications for updates of certain learning objects. Consumers do not require a login and password for the portal. They are treated as anonymous users.

- *Authors* produce learning objects and make them available through the portal. For someone to become an author, he or she needs to request a login and password that grants write access to the system. The system makes sure that authors can only modify or delete material that was uploaded by them. Yet, it is also possible that a number of authors choose to collaborate on the same learning object.
- *Administrators* are responsible for general management tasks, such as adding new users or maintaining the subset of the ACM classification tree supported by the system. If need arises, that is, if too many learning objects reside in the same classifier, authors can request the addition of new ACM classifiers.

The MuSofT portal can be seen as a specialized form of content management system (CMS) for multimedia content. Thus, it would have been basically possible to implement it based on top of an existing CMS. Yet, we found that existing CMS did not meet our requirements:

- A CMS usually distinguishes between a development ("production") view and a presentation ("live") view of the content. This requires the authors to have more knowledge of the system, since the development view is typically more complex than the presentation view. Many systems also assume a dedicated editor who approves changes and a corresponding system-specific editorial workflow - the latter not necessarily being identical to the workflow desired by MuSofT.
- A CMS usually supports a fixed set of metadata. It would have been complicated, if not impossible, to integrate the LOM metadata or the hierarchical ACM classification scheme. None of the systems we evaluated, in particular, supported searching along a hierarchy. In addition, we expected that the set of supported metadata might change over time, so we were looking for a more database-centered solution.

Dissatisfied with existing CMS solutions, we chose to implement the MuSofT portal based on the Infolayer system [Pleumann 2003]. The Infolayer is an object-oriented database the schema of which is specified in the form of a UML class diagram possibly annotated with additional constraints specified in the Object Constraint Language (OCL). The Infolayer's database content is accessible via a Web interface that is generated on the fly based on the database schema. Thus, with only a conceptual class diagram resulting from our early considerations of the LOM metadata to support, we already had a working prototype of the portal. We successively improved this prototype by adding to the system so-called *templates* that specified the layout of individual pages by means of HTML. Nearly all of the development work for the portal was done on the level of either the UML model or the HTML templates. Only few features, such as the e-mail notifications or the IMS content package export, required a "real" implementation. Since in particular the set of ACM classifiers used for structuring learning objects is not hard-coded into the system, but maintained in the database itself, the MuSofT portal is easily reused for other areas than Software Engineering: Only a couple of HTML templates specifying the overall layout and the content of the classification scheme would need to be replaced.

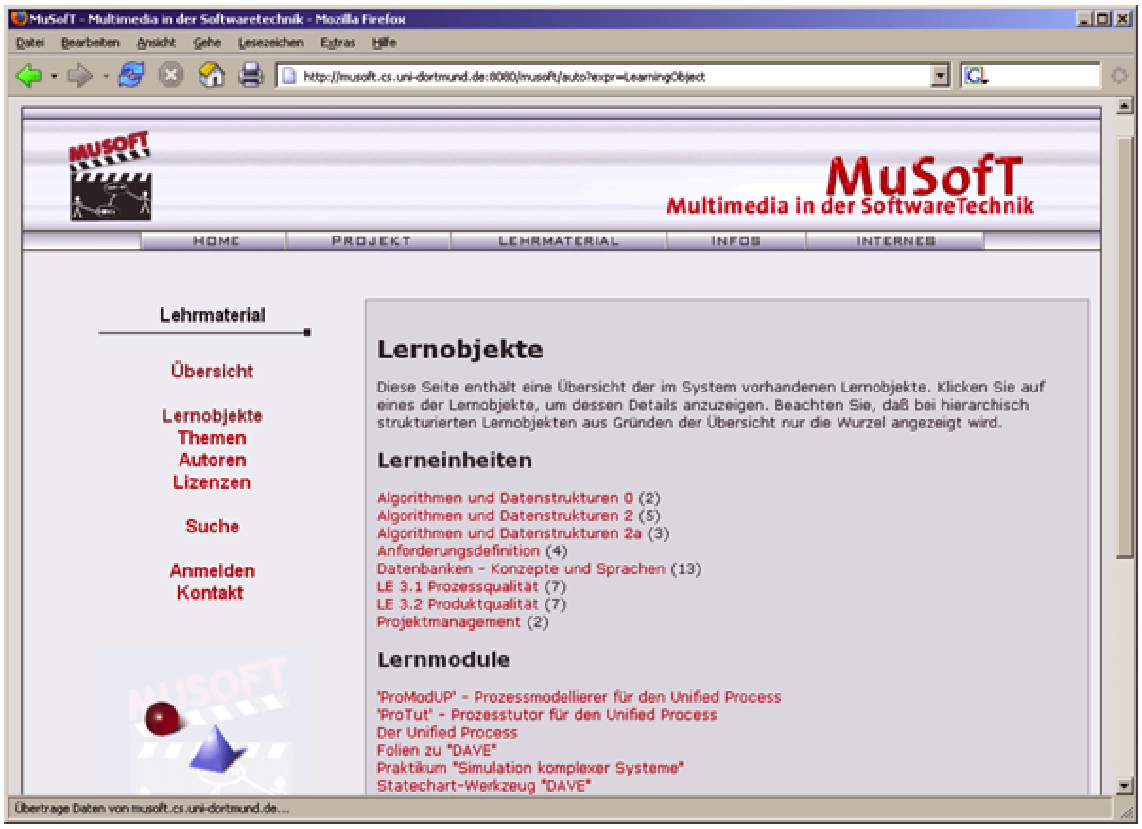

MuSofT - Multimedia in der Softwaretechnik - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://musoft.cs.uni-dortmund.de:8080/musoft/auto?expr=AcmClassifier
MuSofT
Multimedia in der SoftwareTechnik
HOME
PROJEKT
LEHRMATERIAL
INFOS
INTERNES
Lehrmaterial
Übersicht
Lernobjekte
Themen
Autoren
Lizenzen
Suche
Anmelden
Kontakt
Übertrage Daten von musoft.cs.uni-dortmund.de...

Themenübersicht

Auf dieser Seite finden Sie eine Übersicht der Themen, nach denen das Material innerhalb des Systems sortiert ist. Dabei kommt das aus der Literatur bekannte ACM-Klassifikationsschema bzw. eine Untermenge desselben zum Einsatz. Klicken Sie auf eines der Themen, um die Lehreinheiten zu sehen, die diesem zugeordnet sind. Die Zahlen in Klammern geben die Anzahl der Lernobjekte an, die diesem Knoten oder einem seiner Unterknoten zugeordnet sind und für Sie verfügbar sind.

- D. Software (90)
  - D.0 GENERAL (1)
  - D.1 PROGRAMMING TECHNIQUES (E) (7)
    - D.1.0 General
    - D.1.1 Applicative (Functional) Programming
    - D.1.2 Automatic Programming
    - D.1.3 Concurrent Programming
    - D.1.4 Sequential Programming
    - D.1.5 Object-oriented Programming (2)
    - D.1.6 Logic Programming
    - D.1.7 Visual Programming
    - D.1.m Miscellaneous
  - D.2 SOFTWARE ENGINEERING (K.6.3) (76)
    - D.2.0 General (7)
    - D.2.1 Requirements/Specifications (14)
    - D.2.2 Design Tools and Techniques (5)
    - D.2.3 Coding Tools and Techniques (2)

Figure 6 - ACM classification tree in the MuSoft portal


MuSofT - Multimedia in der Softwaretechnik - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://musoft.cs.uni-dortmund.de:8080/musoft/auto?expr=LearningObject
MuSofT
Multimedia in der SoftwareTechnik
HOME PROJEKT LEHRMATERIAL INFOS INTERNES
Lehrmaterial
Übersicht
Lernobjekte
Themen
Autoren
Lizenzen
Suche
Anmelden
Kontakt
Übertrage Daten von musoft.cs.uni-dortmund.de...

Lernobjekte

Diese Seite enthält eine Übersicht der im System vorhandenen Lernobjekte. Klicken Sie auf eines der Lernobjekte, um dessen Details anzuzeigen. Beachten Sie, daß bei hierarchisch strukturierten Lernobjekten aus Gründen der Übersicht nur die Wurzel angezeigt wird.

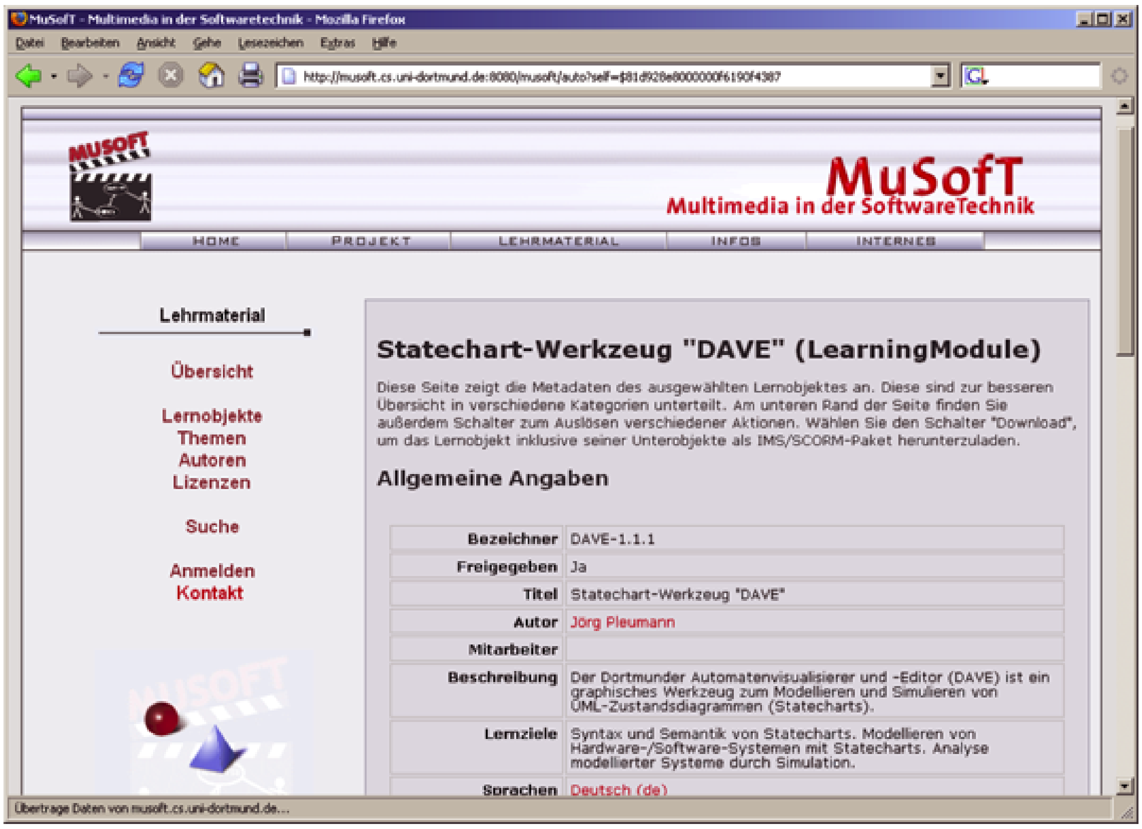
Lerneinheiten

Algorithmen und Datenstrukturen 0 (2)
Algorithmen und Datenstrukturen 2 (5)
Algorithmen und Datenstrukturen 2a (3)
Anforderungsdefinition (4)
Datenbanken - Konzepte und Sprachen (13)
LE 3.1 Prozessqualität (7)
LE 3.2 Produktqualität (7)
Projektmanagement (2)

Lernmodule

'ProModUP' - Prozessmodellierer für den Unified Process
'ProTut' - Prozesstutor für den Unified Process
Der Unified Process
Folien zu "DAVE"
Praktikum "Simulation komplexer Systeme"
Statechart-Werkzeug "DAVE"

Figure 7 - Browsing learning material by granularity in the MuSoft portal


MuSofT - Multimedia in der Softwaretechnik - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://musoft.cs.uni-dortmund.de:8080/musoft/auto?self=$81d928e8000000f6190f4387
MuSofT
Multimedia in der SoftwareTechnik
HOME
PROJEKT
LEHRMATERIAL
INFOS
INTERNES
Lehrmaterial
Übersicht
Lernobjekte
Themen
Autoren
Lizenzen
Suche
Anmelden
Kontakt
Übertrage Daten von musoft.cs.uni-dortmund.de...

Statechart-Werkzeug "DAVE" (LearningModule)

Diese Seite zeigt die Metadaten des ausgewählten Lernobjektes an. Diese sind zur besseren Übersicht in verschiedene Kategorien unterteilt. Am unteren Rand der Seite finden Sie außerdem Schalter zum Auslösen verschiedener Aktionen. Wählen Sie den Schalter "Download", um das Lernobjekt inklusive seiner Unterobjekte als IMS/SCORM-Paket herunterzuladen.

Allgemeine Angaben

Bezeichner	DAVE-1.1.1
Freigegeben	Ja
Titel	Statechart-Werkzeug "DAVE"
Autor	Jörg Pleumann
Mitarbeiter	
Beschreibung	Der Dortmunder Automatenvisualisierer und -Editor (DAVE) ist ein graphisches Werkzeug zum Modellieren und Simulieren von UML-Zustandsdiagrammen (Statecharts).
Lernziele	Syntax und Semantik von Statecharts. Modellieren von Hardware-/Software-Systemen mit Statecharts. Analyse modellierter Systeme durch Simulation.
Sprachen	Deutsch (de)

Figure 8 - Annotating a learning object with LOM metadata in the MuSoft portal

5 Conclusions and Future Perspectives

The MuSofT project was the first joint research project in Germany with a special emphasis on investigating and building educational content and supporting tools for teaching Software Engineering topics. Based on their background, the project partners focused on a subset of Software Engineering topics and created new lecture units by using e.g. multimedia techniques like videos or animations.

A main achievement of the project is the MuSofT portal, which offers a unified access to all MuSofT teaching material. The newly developed MuSofT open content license ensures that the material can be used, adapted, and restored by any lecturer without running into copyright problems. Currently, the portal offers over 250 different learning objects in the field of Software Engineering. The material covers large parts of a typical undergraduate Software Engineering education (e.g., introduction to databases, data structures, and structured development) as well as specialized topics and tools. All material has been developed for and tried in actual lectures. The resulting learning objects are richly annotated with metadata. Interest in the material has been high: overall more than 15,000 visits to individual learning objects contained in the database have been registered, with a total of several thousand downloads so far. To our knowledge, musoft.org thus currently forms the largest public offering of free Software Engineering lecture material in German.

Nevertheless, the MuSofT project could only be a starting point. Due to the limited number of project partners, only a few Software Engineering topics could be treated in depth within the project. Follow-up projects and Germany-wide initiatives are needed to maintain and fill the MuSofT portal in the future with additional material. Only in this case, the MuSofT portal will become THE central source of teaching material for Software Engineering courses. In order to achieve this, we are currently following two routes: the first is to get MuSofT, its tools and its possibilities for the Software Engineering community in Germany better known in this community by changing the attitude from "well, we have heard that there is something going on" to "what are the MuSofT-tools, where are they available, and how can I contribute?". This approach has to fight against a typical attitude in teaching that is very close to the *not invented here syndrome* we all know from the practical obstacles reusing software, which is somewhat counterintuitive, given the ubiquitous complaints on lack of material for teaching. The second route is to attract additional funding for broadening the approach, in particular for extending and consolidating the infrastructure.

Further information on the MuSofT project and its results can be found in [Alfert 2003a, Alfert 2003b, Doberkat 2002a, Doberkat 2002b, Doberkat 2004].

Acknowledgments

The MuSofT project has been conducted by Ernst-Erich Doberkat and Gregor Engels. Both have enjoyed the successful work in the project a lot and would like to thank all project partners and their groups for the excellent cooperation. Klaus Alfert and later Corina Kopka were in charge of the daily chores in project management, Stefan Dißmann not only provided moral support but also gave very practical advice whenever we needed it. Christiane Dusch, Ira Knapke and Andreas Wolfrum of UVM gave legal advice and rendered the open content license a reality. Last but not least we want to thank Sigrid Metz-Göckel and her group for her work which helped rendering MuSofT an environment enjoyable also for female students.

References

(1) [ACM 2005] Association for Computing Machinery: The ACM Computing Classification System, 1998 version. http://www.acm.org/class (last check 2005/08/23)

(2) [ADL 2004] Advanced Distributed Learning (ADL): Sharable Content Object Reference Model (SCORM®) 2004, 2nd Edition, Overview, 2004

(3) [Alfert 2003a] Alfert, Klaus; Doberkat, Ernst-Erich; Engels, Gregor: MuSofT: Multimedia in der Softwaretechnik. In: Bode, A.; Desel, J.; Rathmayer, S.; Wessner, M. (eds.): DeLFI 2003, Tagungsband der 1. e-Learning Fachtagung Informatik, Garching, LNI-37, 2003, pages 115-119, Gesellschaft für Informatik

(4) [Alfert 2003b] Alfert, Klaus; Doberkat, Ernst-Erich; Engels, Gregor; Lohmann, Marc; Magenheim, Johannes; Schürr, Andy: MuSofT: Multimedia in der Softwaretechnik. In: Siedersleben, J.; Weber-Wulff, D. (eds.): SEUH 8 Software Engineering im Unterricht der Hochschulen, Berlin 2003, pages 70 - 80. Heidelberg, d Punkt-Verlag, Februar 2003.

(5) [Allen 2002] Allen, Eric; Cartwright, Robert; Stoler, Brian: DrJava - A Lightweight Pedagogic Environment for Java. Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. ACM Press, 2002.

(6) [Aschenbrenner 2003a] Aschenbrenner, Peter: Eine Lerneinheit für die multimediale Lehre von Algorithmen und Datenstrukturen. In: Bode, Arndt; Desel, Jörg; Ratmayer, Sabine, Wessner, Martin (Herausgeber): DeLFI 2003, Proceeding der 1. e-Learning Fachtagung Informatik. Number P-37 of Lecture Notes in Informatics, pages 412-421. 2003.

(7) [Aschenbrenner 2003b] Aschenbrenner, Peter; Schürr, Andy: Generating Interactive Animations from Visual Specifications. In: Proceedings of 2003 IEEE Symposium on Human Centric Computing Languages and Environments. 2003, Pages 169-176. Auckland, New Zealand

(8) [Doberkat 2002a] Doberkat, Ernst-Erich; Engels, Gregor: Multimedia in der Informatik-Lehre. In: Schubert, S.; Reusch, B.; Jesse, N. (eds.): GI Jahrestagung, 2002, LNI 19, pages 377 - 384, Gesellschaft für Informatik

(9) [Doberkat 2002b] Doberkat, Ernst-Erich; Engels, Gregor: Multimedia in der Softwaretechnik. In: Informatik Forschung und Entwicklung. 2002, 17(1):41-44.

(10) [Doberkat 2004] Doberkat, Ernst-Erich, Engels, Gregor; Kopka, Corina (eds.): Abschlussbericht des Projektes MuSofT "Multimedia in der SoftwareTechnik". Technical Report. Chair for Software Technology. University of Dortmund. 2004. MuSofT Report No. 5.

(11) [Frankel 2003] Frankel, David S.: Model Driven Architecture - Applying MDA to Enterprise Computing. Wiley. 2003.

(12) [Fronk 2004] Fronk, Alexander; Pleumann, Jörg: Relationenalgebraische Analyse von Petrinetzen - Konzepte und Implementierung. In: Kindler, Ekkart (Hrsg.): Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets. Technical Report. University of Paderborn. 2004, Pages 61-68.

(13) [Jacobson 1998] Jacobson, Ivar; Booch, Grady; Rumbough, James: The Unified Software Development Process. Addision Wesley. Reading Massachusetts, 1999.

(14) [Jaeger 2003] Jaeger, Till; Metzger, Axel: Open Content-Lizenzen nach deutschem Recht. In: MultiMedia und Recht, 7/2003. p 431-438

(15) [Harel 1987] Harel, David: Statecharts - A Visual Formalism for Complex Systems. Science of Computer Programming. 1987, 8(3): p 231-274.

(16) [Hendrix 2004] Hendrix, Dean; Cross, James; Borowski; Larry: An Extensible Framework for Providing Dynamic Data Structure Visualizations in a Lightweight IDE. Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. ACM Press, 2004.

(17) [Hodel 1997] Hodel, Werner: Mississippi Queen, Goldsieber Spielverlag 1997.

(18) [IEEE 2002] IEEE: Draft Standard for Learning Object Metadata, July 2002

(19) [IMS 2003] IMS Global Learning Consortium, Inc.: IMS Content Packaging - Version 1.1.3 Final Specification, 2003, http://www.imsglobal.org/content/packaging/index.html (last check 2005/01/27)

(20) [Kamphans 2004a] Kamphans, Marion; Metz-Göckel, Sigrid, Tigges, Anja; Drag, Anna; Schröder, Ellen: Evaluation des Editors DAVE in der informatischen Hochschullehre. Technical Report. Chair for Software Technology. University of Dortmund. 2004. MuSofT Report No. 6.

(21) [Kamphans 2004b] Kamphans, Marion; Metz-Göckel, Siegrid; Schötteldreier, Aira; Drag, Anna: Der Unified Process im Test. Evaluationsergebnisse zum Einsatz des UP in der Informatik-Lehre. Technical Report. Chair for Software Technology. University of Dortmund. 2004. MuSofT Report No. 7.

(22) [Kelter 2002] Kelter, Udo: Versions- und Konfigurationsmanagement in der Ausbildung in praktischer Informatik. Softwaretechnik-Trends, 2002, 22(1): p 26-27.

(23) [Kölling 2003] Kölling, Michael; Quig, Bruce; Patterson, Andrew; Rosenberg, John: The BlueJ System and its Pedagogy. Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology 13(4), 2003.

(24) [Kopka 2004] Kopka, Corina; Schmedding, Doris; Schröder, Jens: Der Unified Process im Grundstudium -Didaktische Konzeption, Einsatz von Lernmodulen und Erfahrungen. In: Engels, Gregor; Seehusen, Silke; (eds.): Proceedings of the 2nd German e-Learning Conference for Computer Science (DeLFI). Paderborn, Gesellschaft für Informatik, 2004, LNI-52, pages 127-138

(25) [Nickel 2000] Nickel, Ulrich; Niere, Jörg; Zündorf, Albert: Tool demonstration - The FUJABA environment. Proceedings of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland. ACM Press, 2000.

(26) [OMG 2003] Object Management Group: The Unified Modeling Language (UML) Specification 1.5. Technical Report. http://www.omg.org/cgi-bin/doc?formal/03-03-01. 2003. (last check 2005/01/27)

(27) [Pleumann 2003] Pleumann, Jörg; Haustein, Stefan: A Model Driven Runtime Environment for Web Applications. In: Stevens, Perdita; Whittle, Jon; Booch, Grady (eds.): Proceedings of the 6th International UML Conference. San Francisco. Number 2863 of Lecture Notes in Computer Science, Springer. 2003, pages 190-204.

(28) [Pleumann 2004] Pleumann, Jörg: Erfahrungen mit dem multimedialen didaktischen Modellierungswerkzeug DAVE. In: Engels, Gregor; Seehusen, Silke (eds.): Proceedings of the 2nd German e-Learning Conference for Computer Science (DeLFI). Gesellschaft für Informatik, 2004, Paderborn, LNI-52, pages 55-66.

(29) [Standish 1995] The Standish Group Report: CHAOS. 1995

---

[1] The MuSofT project has been funded by BMBF in the programme “Neue Medien in der Bildung” between 2001 and 2004 (grant 08NM098)

[2] http://www.imsproject.org/