Managing the development of an e-learning product – Applying software engineering techniques in an academic environment

Karsten Friesen[1,2], Hans Schmitz[1]

[1] Department of Management Accounting and Operations Management, University of Mannheim, D-68131 Mannheim, Germany

[2] Department of Information Systems III, University of Mannheim, D-68131 Mannheim, Germany

Abstract: By drawing an analogy between the development of software and the development of an e-learning product, a set of methods can be re-used. In academic environments however, content experts often have to cope with very limited resources for developing e-learning products, which demands an efficient development process. The paper presents experiences in the application of selected techniques that support the efficiency in an academic environment and discusses proposals on organization, documentation, evaluation, and testing matters on the basis of a real project.

Keywords: learning environments, development process, efficiency, organizing, documentation, testing

1 Introduction

The development of educational software has to deal with the integration of didactic, content specific and technical aspects, which may lead to a very complex project. Such projects are in some ways similar to software development projects because interactive elements have to be implemented in some kind of programming language. Therefore it makes sense to speak of courseware engineering, drawing an analogy to software engineering [KlSt01].

In academic environments however, content experts often have to cope with very limited resources for developing e-learning products, which demands an efficient development process. The achievements of software engineering can be used to improve efficiency, but require resources for management purposes. Our suggestions are based on the experience of two years of development work and use of a product in practice. The next section introduces selected characteristics of developing e-learning products in an academic environment. Section 3 provides a proposal for a straightforward development process. Sections 4 and 5 discuss proposals on documentation, evaluation, and testing that support the suggested approach. The paper concludes with a review of some experiences with the implementation of the process in one of our own projects.

2 Characteristics of developing e-learning products in an academic environment

The limitation of resources is the most important characteristic of developing e-learning products in an academic environment. Thus it is important to concentrate the project's resources on the development of elements that provide unique functionality. Several modules, for example those that provide communication, presentation or database functions, should be implemented by off-the-shelf software components. This design rule leads to a heterogeneous product architecture that incorporates elements of different origin. If one has to deal with very limited resources, the usage of open source components can help to improve the efficiency of the project. For the same reasons the use of development tools often has to be restricted to open source products.

Concerning personnel, limited resources mean that there are only a few, if at all, full time developers. The project staff usually consists of content and software developers with different skills and different levels of involvement. This can often result in communication and coordination problems that can be seen as massive risks to the overall success of the development project.

Because of the big effort it takes to develop an e-learning product, such projects are often a cooperation of different institutions. This leads to a multi-site development with higher complexity, and it often increases the heterogeneity of the major target group, the students. Nevertheless it is very important to be able to cooperate with different instructors, also from outside the project. It enriches the product's contents and didactic ideas.

In an academic environment, e-learning products are in many cases linked to certain classroom courses. Thus the schedule of courses is an important influencing factor for the development process.

We assume in the following that the e-learning product consists of several HTML based, case-driven learning units using some Java applets or applications and multimedia data, such as MPEG films or animations produced using some kind of authoring system.

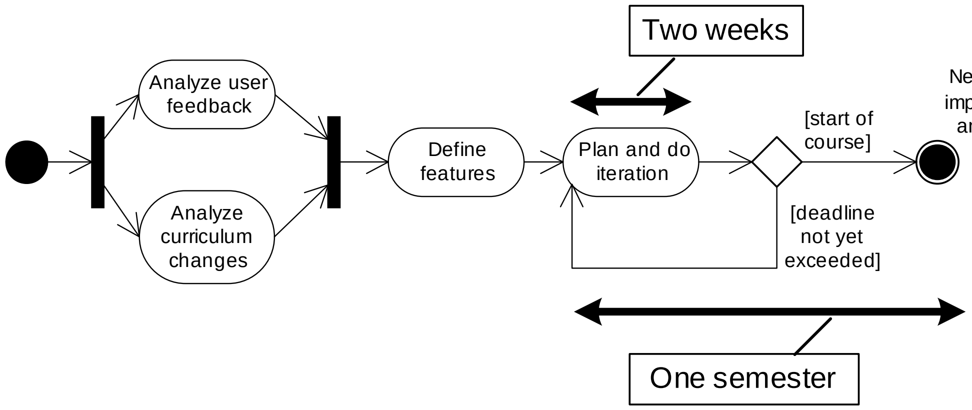

Figure 1: Development process workflow (overview)

3 Basic structure of a straightforward development process

The management of software development has to deal with objectives concerning schedule, features, quality and costs. Although all these aspects have to be balanced in a project, usually one dimension dominates the management process. The major alternatives are a feature driven and a schedule driven project.

The schedule driven approach is generally chosen to guarantee release dates for the customers. The close relation between classroom courses and e-learning products in an academic environment is the major reason for suggesting a schedule driven process. The process is based on well-defined release cycles with one major release for students' use every semester. A second reason for longer release cycles lies in the need to integrate components from outside the project. The integration of updates, e.g. of open-source tools, can be managed efficiently in this period of time.

To actually manage the development process, a "straightforward" approach seems to be appropriate because of the resource restrictions mentioned in the previous section. There are numerous suggestions for software engineering processes. The Rational Unified Process [Kruc00] is an up-to-date example for these suggestions. Unfortunately, the demands for training are high if you want to implement such a process as a whole. Nevertheless it is important to notice the ideas concerning software best practices that are incorporated in such approaches. The Rational Unified Process, for example, is based on the assumption that iterative development, requirements management, component-based architectures, visual modeling, continuous verification of software quality and controlling of software changes are key success factors for the development of software. Taking into account the positive experiences with iterative approaches in successful software companies [CuSe97], the primary idea is to implement an iterative development process for e-learning products.

Thus a semester release cycle is subdivided into several iterations of equal length. Our suggestion is that the smallest building blocks of the project schedule are periods of fourteen days. The length of the iterations is short enough to achieve an appropriate coordination and it is long enough to allow substantial contributions even from part-time developers. Regular meetings at the beginning of all iterations should be used to coordinate the work of the different contributors. These include a review of the current results and the detailed planning of the next two weeks. Between the meetings communication can be provided via a mailing list. If the number of contributors does not exceed approximately ten persons, the communication within the group of contributors is still manageable on a "broadcast" basis, so everyone is informed about all changes to the product. External contributors can easily be integrated into such a communication infrastructure.

Figure 1 shows the development workflow of one release cycle. At the beginning, user feedback and/or curriculum changes should be analyzed as major sources for new global features for the next students' release. The release cycle is strictly schedule driven. The length of a release cycle and the iterations may be varied, depending on the circumstances in a specific institution.

During the whole release cycle and all iterations a modified synch-and-stabilize approach [CuSe97] is applied. All contributions should continually be synchronized and stabilized in order to have a deliverable product at any time. A policy to check in only error-free and previously

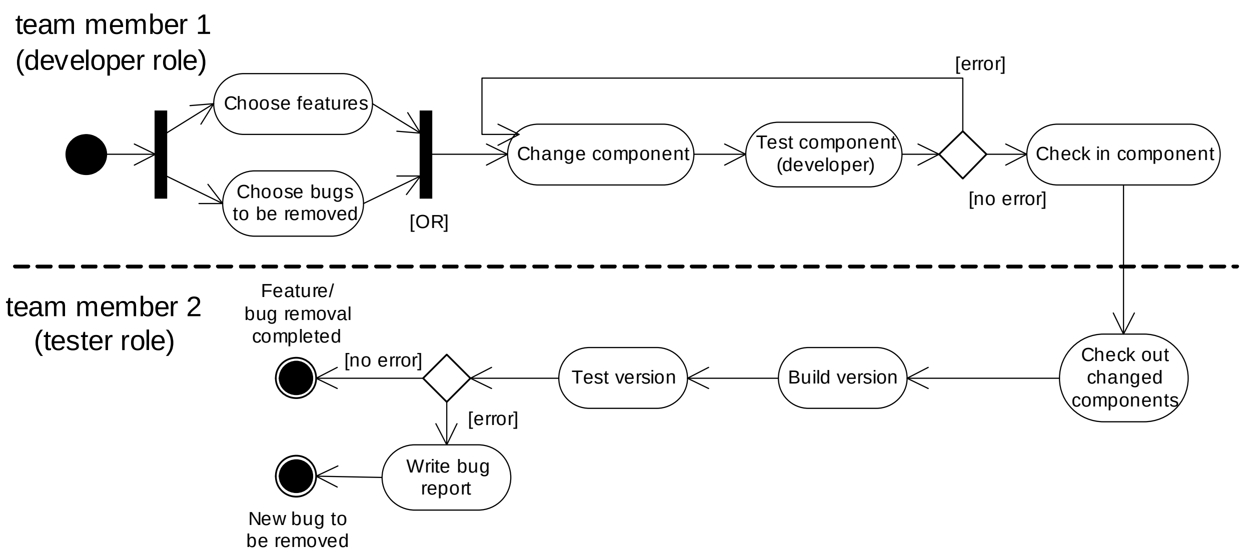

Figure 2: Detailed workflow of the “Plan and do iteration” activity

tested code gives any contributor the chance to obtain the latest version of the product.

Iterations are the basic building blocks of the release cycle. The workflow starts with the determination of the workload for the coming two weeks (see Figure 2). A number of either new features or problems to be solved establish the workload of the next iteration. The iterations include the core workflows of analysis, design and implementation [Kruc00]. As Figure 2 shows, there are two different levels of software quality testing. The developer is responsible for checking in only error free components. This means that there are no compiler errors, that the component interacts accurately with the other existing components and that the feature specification is fulfilled. Therefore all team members are able to check out a running version. All other team members use a current version for their own development work and test the new contributions of their colleagues. If any problems occur, the tester has to write a bug report. The bug report triggers a new development activity.

In e-learning products, different types of components (e.g. multimedia elements, software modules, hypertext) lead to distinct levels of development that can be called *authoring-in-the-small* and *authoring-in-the-large* [KlSt01]. Authoring-in-the-small deals with the development of building blocks for courses, whereas authoring-in-the-large implements whole courses. The suggested straightforward approach is designed to coordinate these two levels of development. Iterations deal with authoring-in-the-small whereas authoring-in-the-large is the global task of every release cycle.

The approach implements a process of continuous improvement of a product, not only the initial release cycle. Documentation, testing, and evaluation are vital to project success and provide key functions in the development process. Therefore they are described in the next two sections in detail.

4 Using documentation, communication, and coordination tools

Looking at the heterogeneity of the contributors and the different levels of involvement, maintaining a proper documentation is considered most important to facilitate communication. UML class and use case diagrams help the content developers to build up a consistent set of basic sample data and give them a better understanding of the premature applications or applets for test and debugging activities.

Another crucial point is the proper documentation of the Java code written within the scope of the project. As mentioned earlier, contributors (students, instructors, developers) naturally have different skills and different levels of involvement. Therefore, it makes sense to have a full-time chief developer who implements, maintains and knows the essential core of the applications whereas the implementation of lesser vital and/or well-separable parts of the product (e. g., user interface parts, test routines etc.) may be delegated to part-time or student developers who only need to know their own code. To assure that everyone knows at least the common interfaces, our suggestion is that any code has to be commented preferably by using the self-documentation capabilities of the respective programming language, e. g., Javadoc tags in

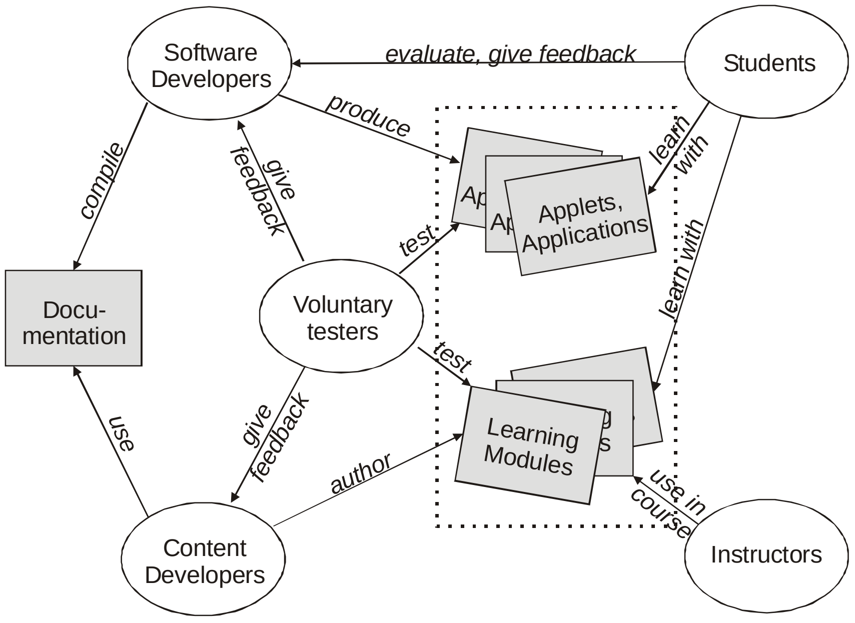

Figure 3: Roles and their interactions in the development of an e-learning product

Java. If we consider that some student developers, especially when they compose their diploma thesis, do not stay a long time in the project, this is the only workable way to access their work after they have quit.

The communication between software and content developers can be supported by a number of web-based documents, which may be stored at the development web server. They comprise authoring aids for tutors and external instructors and a series of case studies with test data and comparative values primarily used for testing. Further aids are a knowledge database, discussion papers, milestones, written records of the regular meetings, the mailing list archives, a web interface for the version control system and more items.

Normally, much can be done on a person-to-person basis, especially when students are involved. So, it is not always necessary and reasonable to use a powerful project management system. However, we recommend to utilize a version control system such as CVS to support the synch-and-stabilize approach mentioned in the previous section and to document the progress of the project.

5 Integrated testing and evaluation for requirements management

During the development process, the prototypes of the learning environment undergo continuous testing in many ways. Since content developers already need parts of the learning system to enter the data of their scenarios, while its expansion is still work in progress, they have to use an early, possibly not fully operative version of the system. In doing so, they check both the results of the operations and the performance of the user interface. To facilitate the testing procedure for every contributor and voluntary tester, appropriate “desk check” scenarios with comparative values should be available for them. Thus the developers obtain a continuous feedback already during the development process. Figure 3 outlines an example of the interactions between the various roles of the development process.

A simple web-based bug-reporting tool may handle the registration of bugs. If a tester discovers a flaw, he gives a short description, the responsible person and a priority from his point of view. The tool forwards the problem report to the appropriate persons who can care about the debugging. The reporting tester checks in regular intervals if the problem is solved and, if so, he changes the state of the report to “cleared”. It is quite normal that one receives approximately 10 bug reports and suggestions for improvement per month; in peak periods, especially when a major summer or winter semester release is to be finished, substantially more.

When complex models are implemented and debugged, it is not an uncommon case that modifications in one point impose unintended side effects at several other locations which are difficult to detect. Those side-effects may result in a partially wrong behavior of the e-learning

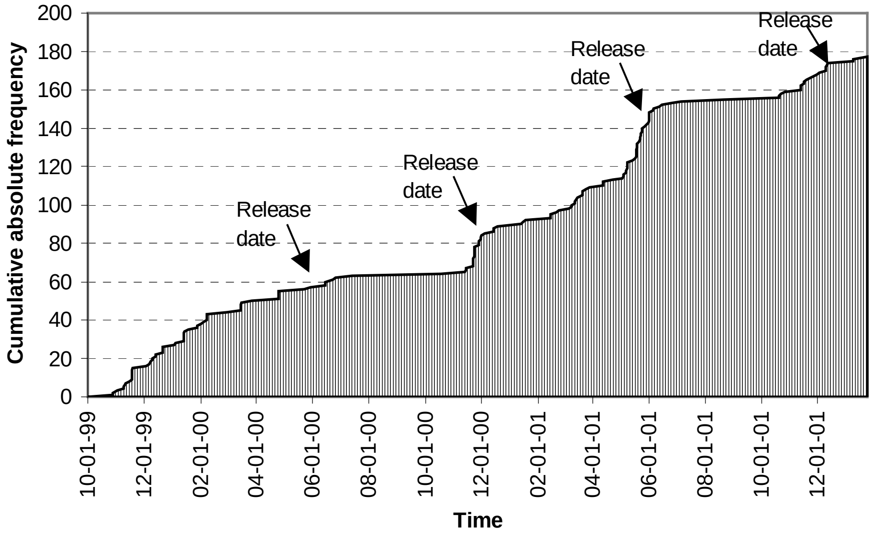

Figure 4: Incidence of bug reports in the Joker project. Bug reports accumulate near the release dates

product even if the students do their exercises correctly. If the e-learning product is in such a way complex, one should seriously consider automated testing to remedy this drawback. That way, common use cases and exercises can be played through regularly to ensure anytime to have a product that delivers exercises which can be understood by students.

Further feedback can be obtained from web based polls when the product is employed in practice in order to generate new requirements and priorities for a forthcoming release. Besides the usual fields that query numeric scorings, suggestions for improvement can be obtained through free-text fields.

6 Building a real product

All the ideas mentioned above were applied in one of our own e-learning projects, "Joker". It may be seen as an exploration tool for students in cost accounting and comprises a fully-fledged cost accounting system that can be used to study various scenarios posed in learning modules. So, Joker consists of three major parts:

- The actual exploration tool as a core application, which covers a wide range of cost accounting functionality
- The data of model enterprises which are processed by the core application
- Various learning modules which introduce numerous aspects of cost accounting. In 2001 there were seven modules available (about 10-15 hours of learning)

The current size of the Java source code totals to approximately 3 MB, compared to the learning modules with more than 100 MB due to the large amount of multimedia content.

The project team consisted of approximately ten members, including students, instructors and full time developers. The contributors all worked at different locations, at home or in their office. As Figure 4 indicates, one major release cycle ends after six months of development. All iterations started with a face-to-face meeting of about half an hour to discuss the project's progress and set the new goals for the coming 14 days. During the iterations the CVS version control system was used, to support the coordination between developers and testers (see Figure 2).

From the beginning of the development process of the exploration tool, it was considered most important to have a common terminology for both the content developers and software developers. The analysis model of Joker is described as a class diagram, which consists of several sub diagrams. It is modeled using the Unified Modeling Language [OMG00] and defines more than 45 persistent Java classes for an appropriate representation of a medium-sized enterprise. It defines basically the core capabilities of the application without consideration of user interface aspects and integration of external software. We experienced that in practice, at least for our project, a few diagrams already suffice. Their use should be confined to support the communication between the contributors.

The version control system was engaged to support the simultaneous development of both the Java application and the learning modules. Content developers used an early version of the product and tested it while they entered the data of a model enterprise. So, the roles shown in figure 3 coincided. Testers reported the bugs they found using a web based tool. Figure 4 illustrates the usage of this tool during the development of Joker. One recognizes that our students usually started to use it approximately from the middle of the semester (June and December), which determined the last-possible release dates and explains the increased debugging and testing activity.

We implemented a test suite for automated testing of the product to remedy the drawback of unintended side effects. Our main objective was to let the test suite prove and verify that the product is functional at least for the scenarios and exercises posed in the learning modules. Thus the solutions of the learning modules were always understandable by the students.

The several hundred users evaluated various parts of the product. Moreover, Joker was subject of one major evaluation study performed by psychologists so far, which gave us valuable information regarding the learning success of the students using Joker, the ergonomics of the user interface, the acceptance of new media in general and a large amount of additional information. While the students' participation in the polls was voluntary, the participation in the evaluation study was not. Hence it took even those students into account who dislike the course and who are only motivated because it was a required course. Our opinion is that we got more substantial results because we consider volunteers to be positively biased towards the subject of the course.

The evaluation study was an integral part of the project. As to the right point of time, we recommend on the basis of our experience to undertake the evaluation in a preferably early stadium of the project to avoid unintended aberrations and to get ideas for improvement that can still be implemented in the remaining time of the project. As banal as it seems it requires the willingness of the developers to evaluate an early version of the product, perhaps with only a few features implemented and an incomplete user interface.

7 Summary

Production of cognitive, interactive content is more than just writing down a good "story". It requires a sufficient cooperation of the parties involved in the developing process. The situation often is even worse in an academic environment where resources are limited. Our experiences so far show that "straightforward" versions of software engineering techniques can improve project efficiency. We use an iterative, schedule driven method, which includes a synch-and-stabilize approach, supported by a version control system. This improves multisite development in a heterogeneous development team. Documentation follows the specific coordination and communication needs of the project. The number of documents is limited to the bare essentials, but the documents remaining are used intensively. Testing and evaluation are identified as very important means to capture requirements. Detailed user feedback is needed to set priorities for new features. Evaluation activities can be substantially improved by cooperations with experts with a strong didactic background.

8 References

[CuSe97] *Cusumano, M. A.; Selby, R. W.*: How Microsoft Builds Software. In: Communications of the ACM 40 (1997) 6, 53-61.

[KlSt01] *Klein, M.; Stucky, W.*: Ein Vorgehensmodell zur Erstellung virtueller Bildungsinhalte. In: Wirtschaftsinformatik 43 (2001) 1, 35-45.

[Kruc00] *Kruchten, P.*: The Rational Unified Process. Addison-Wesley, Boston et al. 2000.

[OMG00] *Object Management Group:* OMG Unified Object Modeling Language Specification (Version 1.3), 2000. Available from
ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf