An Object-Oriented Software Life Cycle of an Intelligent Tutoring System

Maria Virvou, Victoria Tsiriga

Department of Informatics,
University of Piraeus,
Piraeus 18534, Greece
E-mail: {mvirvou | vtsir}@unipi.gr

Abstract. This paper describes the application of an object-oriented model of life cycle in the development of an Intelligent Tutoring System for the domain of Algebraic powers. The system is called EasyMath and has been developed following the Rational Unified Process, an object-oriented method that supports multiple recursions of the phases of the software life cycle. This has been enriched to include empirical studies and evaluations appropriate for knowledge-based educational software.

1 Introduction

In the recent years, there is a growing need for computer technology to be used in the environment of real school and/or higher education classrooms. However, educational software has often been criticised that it has not been designed to meet the needs of real classrooms. For example, Gilbert (1999) analyses the Teaching and Learning Technology Programme (TLTP) evaluation report by Higher Education Funding Council for England. Among other things, he notes that the programme seriously underestimated the complexity of designing materials that could be considered, in any sense, “intelligent” and that there has been a serious lack of evaluation. On the other hand, Intelligent Tutoring Systems (ITSs) incorporate intelligent
features but have often been criticised (Boyle, 1997) as limited research projects that cannot be used in real world settings.

The above criticisms show that there is a need to develop ITSs in a more systematic way, according to software engineering methods. A solution to this problem may be the involvement of school teachers and students in the development of an ITS (Virvou & Tsiriga, 1999a) and the application of evaluation methods on every stage of the system's life cycle, to ensure usability and learning effects. The idea of involving end users (such as students) and experts (such as human tutors) is not new in knowledge and/or software engineering. However, their involvement is often neglected, especially in later stages of the development, resulting in systems which are not very usable.

For the purpose of this research we have developed a multimedia educational system aimed at teaching students the domain of Algebraic powers. The system is called EasyMath and it incorporates intelligence. EasyMath has been designed to be used by students and/or teachers. Students may use EasyMath to read theoretical lessons about Algebraic powers and solve exercises. Teachers can use EasyMath as an assisting tool for the creation of new exercises, which can serve as student tests.

In order to ensure the usability of the ITS in real school classrooms, EasyMath’s model of life cycle has been based on the Rational Unified Process (Booch et al., 1997; Kruchten, 1999), which is an object oriented process that suggests multiple iterations of the software development phases. Indeed, given the complexity matters involving the development of an ITS, there is a great difficulty in following a classical software engineering life cycle such as the waterfall model (Sommerville, 1992). This kind of model does not seem suitable, since it does not allow users to test the system until it is complete.

Traditionally, an ITS would be regarded as a knowledge-based system, in which case knowledge engineering advocates the creation of prototypes in the process of the development. An iterative approach, as the one in the spiral model, would make it easier to accommodate tactical changes in requirements, feature or schedule (Boehm, 1988; 1996). However, we have chosen the Rational Unified Process as the model of life cycle of EasyMath for two reasons. Firstly, it supports multiple iterations of the software development phases, a crucial matter for systems that should pursue usability and learning purposes in an integrated way. Secondly, it is an object-oriented methodology, which suits better the development of graphical user environments, such as EasyMath.

Using the Rational Unified Process the software life cycle is broken into cycles, each cycle working on a new generation of the product. The process divides one development cycle in four consecutive phases: the inception, the elaboration, the construction and the transition phase.
The remaining sections of this document are named after the phases of the life cycle. In each of them we describe the work done. The only phase that is not discussed is the transition phase, since the system has not been through this phase yet; such a phase would require the distribution of the system in a large number of schools. This is not possible without the collaboration of a national organisation concerned with the syllabus of state schools. Finally we give the conclusions of this research.

2 Inception Phase

One of the main aims of this phase was the primary specification of the requirements of the ITS. However, in the case of knowledge-based educational software we believed that we had to conduct a thorough empirical study that would involve human teachers and students. In particular, the empirical study involved 4 school teachers and their students. Each human tutor taught 2 different classes of the same grade. Each class consisted of 30 students. All students were given the same tests, which were created by the 4 teachers in collaboration. These tests covered all the material taught in Algebraic powers. The results of these tests were analysed by the human tutors, so that as many errors as possible could be categorised. As a result of the analysis, the most common categories of error were identified and the cause of them was diagnosed by the human teachers. For a detailed description of the results of the empirical study, the reader is referred to (Virvou & Tsirigia, 1999b).

A major source of difficulty in developing and delivering successful software is the difference between the user and developer perspectives. Users are focused on the problem domain, where the system's features are the primary concern, whereas developers are focused on the solution domain (Turner et al., 1999). Therefore, while developing software that is meant to be usable in real world situations, it is crucial to describe users' requirements in a way that all participants can comprehend.

In the case of EasyMath, we have used UML in order to capture and describe the requirements of the system. In UML, the requirements on the system's functionality are described in terms of use-cases and actors (Jacobson et al., 1992). Through use-case modelling, the external actors that have interest in the system are modelled along with the functionality they require from the system (the use-cases). The actors and use-cases are modelled with relationships, and have communication associations with each other or are broken down into hierarchies.

Although use-case modelling has received wide interest from the community of object-oriented engineers, it has also been reported to have some problems. For example, Muller et al. (1997) point out that
one disadvantage of the approach is that the produced use-case model usually is written with the software system as a focus of attention, giving too little priority to the end-users. To overcome this difficulty, in the case of EasyMath, the descriptions of the requirements using use-case diagrams was done by the designers of the system in collaboration with human tutors and students.

Figure 1 about here.

An example of a use-case diagram concerning the students’ use of EasyMath, is presented in Figure 1. In this diagram, we present the process of solving exercises using EasyMath. The first step of this process is the creation of a new exercise for the student to solve. The system then shows the exercise and remains idle until the student provides his/her answer. In case of a correct answer, the system confirms the correctness. On the other hand, if a student gives an erroneous answer, EasyMath tries to perform error diagnosis by generating the faulty procedures of solution. If one of them is found to match the student’s answer then EasyMath shows to the user the appropriate kind of advice.

3 Elaboration Phase

The purpose of the elaboration phase is to analyse the problem domain, establish a sound architectural foundation and produce a first prototype system to serve as a basis for the iterative development of the final product. In this stage, architectural decisions have to be made with an understanding of the whole system. In addition, while the process of the development must always accommodate changes, the elaboration phase activities ensure that the architecture, requirements and plans are stable enough, and the risks are sufficiently mitigated (Kruchten, 1999).

In the case of EasyMath, the outcomes of the elaboration phase were the refined use-case diagrams, the description of the system architecture, and a primary executable release of the intelligent tutoring system. The design of the system has been based on the functional requirements of the system, as described in the documentation provided by the inception phase. In order to ensure that the system under development would be useful to real school environment, one of the human tutors also participated in the phase of elaboration. His role was to collaborate with the designers in the processes of refinement of the use-case diagrams developed in the inception phase, and of the development of the prototype system.
3.1 System Architecture

The architecture of EasyMath follows the main line of Intelligent Tutoring Systems architectures. Researchers in the area of ITSs largely agree that there are four major functional components constituting an ITS architecture, namely the Domain Knowledge, the Student Modeller, the Advice Generator and the User Interface (Hartley & Sleeman, 1973; Wenger, 1987; Nwana, 1990).

The domain knowledge of EasyMath contains the explicit representation of the theoretical concepts of algebraic powers. It comprises knowledge about how to solve an exercise concerning algebraic powers in the correct way. Furthermore, it provides the knowledge needed for the dynamic construction of new exercises, which can be used as exercises or examples shown to the user.

The student modelling component is the part of an ITS that manages the explicit assumptions about a student in respect to his/her interaction with the tutor. The student modelling component of EasyMath has been based on the buggy approach, which was first introduced in BUGGY (Brown & Burton, 1978). Since then it has been used in many other systems (Langley et al., 1987; Sleeman et al., 1990; Hoppe, 1994). EasyMath reconstructs the problem-solving process and generates mal-rules from hypothesised faulty solution paths, which are used for the modelling of students’ misconceptions or procedural bugs.

The advisor is the part of an ITS that is responsible for the didactic strategy of the system. The advisor of EasyMath is responsible for providing individualised feedback to students’ answers. Finally, the user interface of EasyMath, is a multimedia user interface, which involves pictures, animations and sounds so that it can attract the student’s interest in the subject.

EasyMath aims at helping human teachers and students in real classroom settings while incorporating intelligence. For example, the module of EasyMath that creates new exercises, is based on the bug library of the student modelling component in order to dynamically construct multiple choice and other exercises concerning algebraic powers. The system can assist the teacher, since it contains knowledge about how to solve an exercise correctly as well as in several faulty ways. In addition, it provides explanations about the students’ errors based on the individual long term student model.

3.2 Evaluation of the Primary Executable Release

As Dix et al. (1993) point out, evaluation is an integral part of the design process and should take place throughout the design phase of the life cycle. Therefore, the primary executable release developed in the elaboration phase was evaluated along the functionality requirements posed in the use-case model. School
teachers as well as students also participated in the stage of the evaluation of the primary executable release of the tutoring system.

The primary executable release evaluation was based on qualitative evaluation methods, such as observation and questionnaires. The construction of the questionnaires was based on a set of “usability” heuristics (Nielsen, 1994), which have been proposed as a technique for the evaluation of interfaces. In the case of EasyMath we have used them in order to evaluate the primary executable release, in terms of its friendliness and usability. The heuristic approach to evaluation was chosen since it seems to focus evaluators’ attention as they work their way through the system.

In particular, in this phase of the evaluation, 2 teachers and 15 students were asked to operate with the primary executable release of EasyMath in order to complete several predefined tasks. One of the system designers noted down the difficulties that students phased while they worked with EasyMath. These difficulties were analysed and used in order to refine the system. Moreover, after the interaction with EasyMath, both teachers and students were asked to complete a questionnaire, concerning the usability and friendliness of the system. As already mentioned, this questionnaire was constructed based on a set of “usability heuristics” and contained questions such as:

1. Were instructions for use of the system visible or easily retrievable whenever appropriate?
2. Were the symbols and the words used in the different components of EasyMath consistent?
3. Are you familiar with the symbolic representation of the algebraic powers used in the exercises?
4. Is the amount of different forms of exercises (multiple choice, game, student’s answer) adequate?
5. Do you believe that the presentation of the exercises is suitable?

Finally, human teachers were also asked to make additional comments about possible enhancements that would make the system more helpful and useful in real school settings.

The comments made at this stage of development of the tutoring system formed the basis for the refinement of the functionality requirements of EasyMath. For example, the human teachers suggested the addition of a new type of exercise apart from classical multiple choice and solving exercises mode that would enhance the motivation of students. As a result, a game playing mode was added to the system. In this mode the students are given the opportunity to learn more about the domain of algebraic powers while solving a puzzle.
Furthermore, the evaluation of the primary executable release provided significant information concerning the user interface design and development of the system, so that it could be more helpful to teachers and students in real school settings. For example, one comment made by the end-users concerned the multiple choice question answering mode. In particular, in multiple choice tests, human tutors would prefer students to be shown more than one questions in one page. The system originally showed one question in each page and provided a result report after a set of ten questions (pages). Human teachers claimed that in that way students would have forgotten what answers they had given in each question before they were shown the results.

4 Construction Phase

During the construction phase, all remaining components and application features are developed and integrated into the product. In addition, the resulting software is thoroughly tested among all aspects. The outcome of the construction phase is a product ready to be used by its end-users.

In the construction phase, the prototype system of EasyMath that was developed in the previous phase was extended, leading to a second executable release of the tutoring system. The product that resulted from this phase was also tested thoroughly in terms of its usability and functionality. For the formative evaluation of the second executable release, we used a set of “learning with software” heuristics (Squires & Preece, 1999). These heuristics are an adaptation of “usability heuristics”, so as to relate to socio-constructivist criteria for learning. The reason we used these heuristics was to render the evaluation appropriate for the purposes of educational software. The results of the formative evaluation form the basis for the future development plans concerning EasyMath.

5.1 Evaluation of the "Beta" Release of EasyMath

EasyMath is meant to be used by both teachers and students. Therefore, in the phase of the formative evaluation of EasyMath, 10 school teachers as well as 240 students were involved. Both students and teachers were asked to evaluate EasyMath in terms of the purpose of education. In addition, teachers were asked to evaluate the overall performance of the system and express their opinion about the usability of such an intelligent tutoring system in a real classroom.

In the stage of the students' evaluation of EasyMath, the total of 240 students were separated in two portions. The first 120 students were introduced to EasyMath and then were asked to use it for about one
The students, after interacting with EasyMath were given a questionnaire to complete. The questions included in the questionnaire were related to usability, learning, and integrated issues concerning EasyMath. As for the rest 120 students, they were taught half of the syllabus in algebraic powers without the use of EasyMath and they were given a written test. Then they used EasyMath while being taught the rest of the syllabus. In the end they were given another written test and the grades of their first and second test were compared. The results of this research showed that 46% of the students obtained a better grade in the second test, 43% obtained the same grade and only 11% obtained a lower grade in the second test.

The evaluation questionnaires were constructed based on the "learning with software" heuristics. These heuristics have been suggested as a method for performing predictive evaluation. However, we have used them to construct a formative evaluation questionnaire that would give us insight about the integration of both usability and learning issues.

In the stage of the teachers’ evaluation of EasyMath, they were asked to role play an average student interacting with EasyMath. Next, they were given a questionnaire to fill in. The questionnaire was carefully designed so as to ensure that questions are related to as many of the "learning with software" heuristics as possible. A sample of the questions included is the following:

- Were you satisfied with the responses to students’ errors?
- Was it easy to get familiar to using EasyMath?

Given the results of the overall system evaluation, it is shown that EasyMath seems to be a useful product for schools. The multimedia interface of EasyMath renders the system quite attractive to students. Moreover, the student modelling component of EasyMath provides individualised support to students’ learning. However, the evaluation also showed that there is scope for improving the student modelling component in terms of the ambiguity resolution in multiple hypotheses about errors.

### 6 Conclusions

To produce intelligent educational software, useful in real classrooms, there is a need to ensure that all functionality requirements are met and the knowledge acquisition has been based on real human experts rather than speculations of a knowledge engineer. Finally, the usability of the product has to be evaluated in real world situations. One way to achieve the above goals may be the application of evaluation methods and
the involvement of teachers and students throughout the life cycle of the development of the intelligent tutoring system.

The Rational Unified Process has provided an adequate framework for the necessary multiple iterations of the development process. However, it is necessary to have a further adaptation to the particular needs of knowledge-based educational software. Such needs advocate the inclusion of empirical studies and evaluations that address both learning and usability issues, such as heuristic evaluations.

References


Figure 1: Use Case Diagram concerning the students’ solving exercises process.