Published in *Proceedings of EDMEDIA 2004 – World Conference on Education Multimedia, Hypermedia and Telecommunications*, pp. 1590-1595. June 23-26, 2004, Lugano, Switzerland.

On the Learning in E-Learning

Raimond Reichert
Swiss Centre for Innovations in Learning
University of St Gallen
9000 St Gallen, Switzerland
raimond.reichert@unisg.ch

Werner Hartmann
Technology & Education
Swiss Federal Institute of Technology
8092 Zürich, Switzerland
hartmann@inf.ethz.ch

Abstract: In E-Learning, the emphasis is often on technical aspects, and the reason for using technology – the desired pedagogical added-value – is in danger of being neglected. The emphasis of this paper is on the actual process of learning, the learning in E-Learning, and how this process can be assisted by interactive educational software. We briefly present five high-level quality criteria which good educational software must satisfy. Two examples of such software show that a high degree of interactivity has its unavoidable price. We discuss the tradeoff between interactivity and cost of development, making the point that interactive learning environments can not be developed with general purpose authoring tools or learning management systems, but instead require extensive, domain-specific development effort.

E-Learning: From Interpassivity towards Interactivity

Many educators believe that information and communication technologies fundamentally transform and improve the process of learning. This assumption is not new: Each major step in the evolution of modern technology gave raise to new hopes and promises. However, the new technologies have, in most cases, not had the proclaimed effect. In E-Learning, the emphasis is often on the technologies used and on learning management systems (LMS). This type of software is independent from the content it manages, and can therefore not improve the process of learning by itself. Typically, these programs rarely support more than multiple choice questions, mappings, and quizzes. Such rudimentary forms of "interaction" are technically trivial to implement, but they are more "interpassive" than interactive. To get an added pedagogical value from an LMS, the emphasis must be more on the learning itself rather than on content management and distribution.

It must be noted that there are two kinds of interaction in E-Learning environments which are often confused. The first kind, based on a social notion of interaction, is human-human interaction through technology, also referred to as computer mediated communication. Instances of this type of interaction are text or video chat rooms, discussions forums, application sharing etc. This form of interaction can be supplied by general purpose communication tools typically found in LMS implementations. The second kind of interaction, based on a technical notion of interaction, is human-computer interaction, referred to as interactivity. Examples include educational software, micro-worlds, simulations etc. This form of interaction can not be provided by an LMS itself, but rather requires special purpose development effort.

In the following, we focus on interactive educational software, also referred to as interactive (virtual) learning environments. We believe that its potential is far from exploited. The use of educational software can enrich the process of learning, provided that there is a pedagogically sound concept for its use. To achieve this goal, it is necessary to focus on the process of learning, on the Learning in E-Learning. The primary question has to be how technology can yield an added value from a pedagogical perspective: What can be done with technology that could not be done just as well without it? Figure 1 shows different components of a virtual learning environment. The focus of this paper is on interactive content. "Interpassive" elements such as handouts, animations, or lecture videos belong to the category „E-Distribution" of content; they do not add anything to the actual process of learning which could not be done without computer technology.

Published in *Proceedings of EDMEDIA 2004 – World Conference on Education Multimedia, Hypermedia and Telecommunications*, pp. 1590-1595. June 23-26, 2004, Lugano, Switzerland.

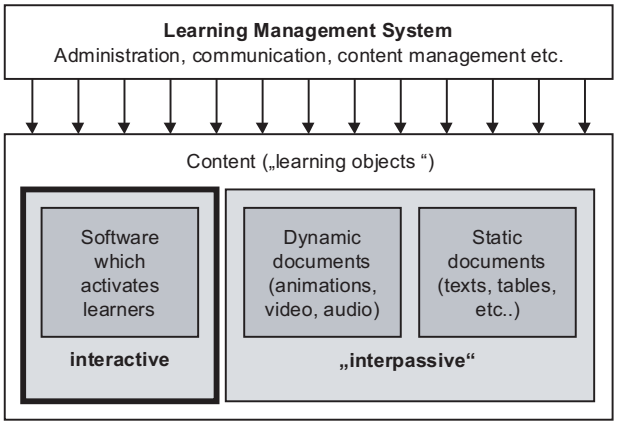

Figure 1: Interactive educational software versus “interpassive” documents

Criteria of “Good” Interactive Educational Software

The quality of educational software is a product of many factors, and there are different quality guidelines. We restrict ourselves to a small number of quality criteria on a high level of abstraction, drawn from a broad range of disciplines: pedagogy, human-computer interaction research, and multimedia design research. We point to a few selected literature references. Our criteria serve as guidelines for the conception and implementation of educational software, even though they do not offer a simple checklist to follow. We do not claim that our criteria are complete, since completeness might well prove to be an elusive goal. Our criteria provide a goal to strive towards: One will not be able to satisfy all criteria equally well with any particular educational software.

Content Based on Fundamental Ideas The production of educational software is expensive. Therefore, long-lived content should be at the center of interactive learning environments. Bruner's concept of „fundamental idea“ is a suitable guiding principle for the selection of long-lived content (Bruner, 1960). Since Bruner himself gave only an intuitive description, but no precise definition of the concept of fundamental idea, we summarize his idea according to Schwill (1994) as follows: A fundamental idea with respect to some domain is a schema for thinking, acting, describing, or explaining which is applicable in different areas, may be demonstrated and taught on every intellectual level, can be clearly observed in the historical development and will be relevant in the longer term, and is related to everyday language and thinking. In other words, fundamental ideas guarantee the selection of content which is cognitively demanding, relevant, and long-lived.

Incorporating Different Cognitive Levels Ideally, educational software offers a broad range of tasks on different cognitive levels. Bloom (1956) developed a taxonomy of six cognitive levels of increasing complexity: knowledge, comprehension, application, analysis, synthesis, and evaluation. Good educational software addresses and emphasizes, in particular, the higher cognitive levels, that is, the levels of analysis and synthesis. It is interesting to note that with popular computer games such as SimCity, this has been the case for a long time.

High Degree of Interactivity A high degree of human-machine interaction characterizes good educational software. By interactivity we do not mean reading a web page or watching an animation, but true interaction between the learner and the software. Laurel (1993) defines this type of interaction as follows: „You either feel involved in the computer representation or you do not. The crucial point is the ability to interact with the representation, and not how often the software feigns communication with you.“

There are different classifications of interactivity levels. We use a model by Schulmeister (2003) which defines six levels of increasing human-computer interaction. Level one means no interaction at all, but only a display of information. Level two lets users navigate through the representation of information. Level three offers multiple representations of the content. On level four, the user can modify parameters of the representation. Additionally, on level five, the user can manipulate the content itself. Level six means the user can create and manipulate objects and watch the system react.

Published in *Proceedings of EDMEDIA 2004 – World Conference on Education Multimedia, Hypermedia and Telecommunications*, pp. 1590-1595. June 23-26, 2004, Lugano, Switzerland.

Only few computer based learning environments satisfy the demand for a high degree of interactivity. One reason is the high cost of development. As Berg (2002) notes in *The Big Questions*, „Highly interactive software using simulation strategies is almost non-existent in higher education. Clearly the cost of developing such software is a barrier.“ Educational software which focuses on fundamental ideas of a domain and consequently addresses different cognitive levels has the potential to amortize the high cost of development, because the pedagogical concept underlying the software will be of long-lived value.

Feedback The software's feedback with respect to the actions of the users can assist their learning process. We roughly define two levels of feedback, „implicit“ and “explicit“. With implicit feedback, the learners must interpret the output that the software produced while they interacted with it. Explicit feedback denotes an automated tutor which takes on the role of the teacher. The tutor points learners to mistakes they make, provides support when the learners stumble or shows them different possible solutions. However, the term „intelligent tutorial system“ is often used when the only feedback is a simple yes-or-no feedback as is the case in multiple choice questions. Explicit feedback implies a feedback which specifically takes into account the user's interactions with the software. This type of differentiated feedback is usually only possible when the domain at hand is either highly structured and can be captured formally, or when the feedback can be generated from a huge database of expert knowledge.

Visualization and Usability It takes time to familiarize oneself with the user interface of any software. Since the user interface is not the subject itself, it should be as self-explanatory as possible. However, as of yet, there is little if any discussion on usability in the E-Learning context, on what could be called the “learnability” aspects of electronic media. However, it should also be noted that there are scientific guidelines to the design of multimedia objects, for example *Multimedia Learning* by Mayer (2001). It is unfortunate that in practice, such guidelines are often overridden by rules of thumb like “animations increase learning”.

Moreover, one should also take into account the so-called „Nintendo generation effect“. Guzdial and Soloway (2002) argue in *Teaching the Nintendo Generation to Program* that the high dropout rates in computer science courses are caused by the educators' outdated view of information and communication technology. Whereas “Hello World“ programs got students excited when computers were still text based, today's „Nintendo generation“ grows up in multimedia environments. Educational software needs to correspond to these multimedia environments and to the students' every day use of computers. Doing so, it extrinsically motivates learners, boosting students' exploration of the content underlying the software.

Two Examples of Interactive Educational Software

To what degree the criteria given above can be taken into account when designing educational software depends on the subject domain. In the following, we present two examples of educational software which satisfy our demanding criteria in different ways and to varying degrees. The examples show that in order to put the emphasis on the Learning in E-Learning, highly content-dependent pedagogical design and software development is indispensable. The first learning environment has been developed in the Technology & Education research group at the Swiss Federal Institute, Zurich. The second example presents a well known application from medical education.

Kara: Introduction to Programming

In today's Information Society, knowledge of the fundamentals of information and communication technology is a key qualification and must become part of general education. The concepts of formalization and programming are at the heart of computer science: Descriptions of processes which are easily understandable by humans must be translated to formally precise descriptions digestible by computers. Teaching these concepts as part of general education is a difficult challenge. In the last thirty years, many educational programming environments were developed with the goal of making the introduction to programming easier. The most prominent example is probably Seymour Papert's Logo based approach where users program a “turtle”.

Here we briefly present the Kara environment for introductory programming (Reichert et al, 2001). The environment's programming mechanism is based on the model of finite state machines. The concept of this type of machine is easily understood, and there are many everyday devices whose logic illustrates finite state machines, such as watches, vending machines, or VCRs. Moreover, state machines can be visualized in a straight-forward

Published in *Proceedings of EDMEDIA 2004 – World Conference on Education Multimedia, Hypermedia and Telecommunications*, pp. 1590-1595. June 23-26, 2004, Lugano, Switzerland.

manner. The environment lets users program the ladybug Kara to solve problems in his world. Even though the world is simple, there are tough problems to be solved, such as the drawing of a Pascal triangle modulo 2 (figure 2).

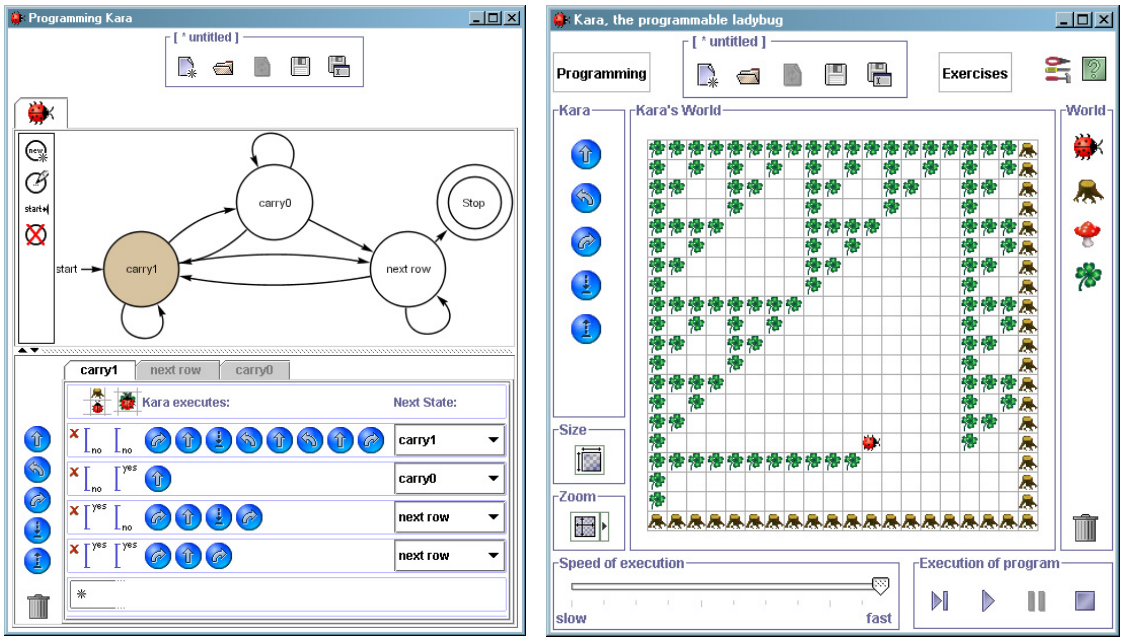

Figure 2: Introduction to programming with Kara

We briefly consider which of our criteria of good educational software the Kara environment satisfies:

Fundamental Ideas The formalization of algorithms, and the translation of algorithms into programs, is a core topic of computer science.

Cognitive Levels The tasks to solve range from easy getting-started tasks to demanding tasks from the theory of computation. When programming, users must evaluate different possible algorithms for their problems, that is, the tasks address the highest level of Bloom's taxonomy.

Degree of Interactivity The environment is highly interactive: Users create their own worlds and write programs which they test against their worlds.

Feedback Bugs in the users' programs become visually obvious during program execution. Users see when the ladybug does not do what they intended it to do. However, there is no explicit feedback which specifically takes into account the users' programs and tells them what causes the bug.

Visualization and Usability Kara is easy to use, and its game-like nature makes it attractive to students. They like the environment's graphical "wrapping", yet are quick to realize that programming is demanding.

It took five man-years to develop the Kara software, most of which was spent implementing and evaluating the software. Since Kara was developed within the context of a framework, much code can be reused for related subjects of computer science.

Interactive Simulated Patient: Active Learning with Virtual Patients

An important part of medical education is the diagnosis of illness. Ideally, the students practice on real patients. However, that is very expensive, and often difficult to realize. The cases of interest may not always be available just when they would be needed in class. Also, it is often not easy to get the patients to talk about their illness in front of students. For these reasons, the project *Interactive Simulated Patient* was initiated in 1990 (Bergin and Fors, 2003).

Published in *Proceedings of EDMEDIA 2004 – World Conference on Education Multimedia, Hypermedia and Telecommunications*, pp. 1590-1595. June 23-26, 2004, Lugano, Switzerland.

The project developed a virtual learning environment for the diagnosis of patients. The system is designed to foster active, problem based learning and to be as realistic as possible. Basically, the system consists of a huge multimedia database of patient data (example in figure 3). The main task for the students is to come to a efficient, cost-effective, and of course correct diagnosis.

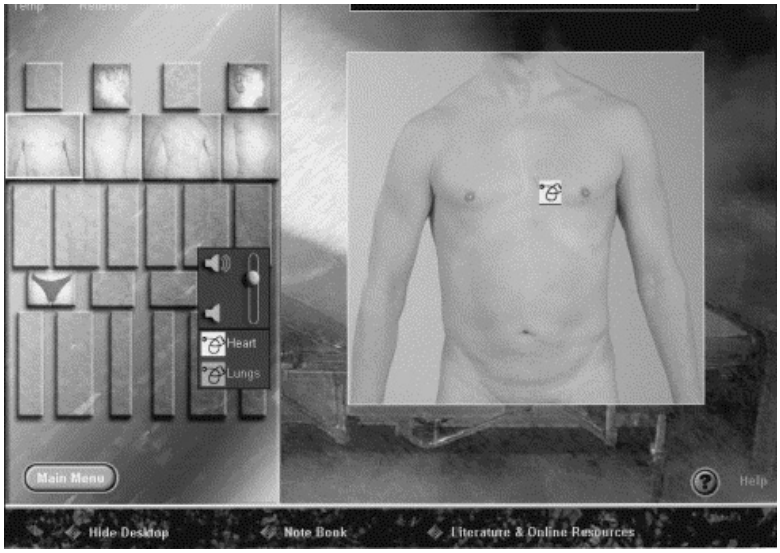

Figure 3: Diagnosis of a virtual patient

Again, we briefly consider which of our criteria of good educational software the Interactive Patient satisfies:

Fundamental Ideas Diagnosis is one of the core tasks of every medical practitioner. The students must do a heuristic search in a loosely structured search space – the patient's data – while trying to minimize the cost of the search and the diagnostic tools used for it.

Cognitive Levels The problems the students must solve require comprehensive medical knowledge. They must analyze the patient's situation, draw their own conclusions from this analysis, and act on it.

Degree of Interactivity Since the search space of patient data encompasses a huge amount of data for each patient, a high degree of interactivity results.

Feedback The system gives an implicit feedback by showing students the correct diagnosis and how to arrive at it with minimal cost.

Visualization and Usability The multimedia data which the students use yield a quite realistic impression of the patients. In evaluations, students graded the *ease-of-use* of the environment a 7.5 average on a scale from 1 to 10.

The quality of this learning environment is ensured by the effort of an interdisciplinary team of over 20 specialists which collected the necessary high-quality patient data over 13 years. The high degree of interactivity is not due to programming – an authoring tool was used to develop the environment – but rather to the encompassing collection of patient data and the experience of the specialists involved in putting it together. This type of work can not be automated, and its results can not be reused for other domains.

Conclusion: The Tradeoff between Interactivity and Cost of Development

General purpose authoring tools and learning management systems are not enough to satisfy the high requirements set forth in the criteria given in this paper – neither LMS nor authoring tools can improve the process of learning per se. The desire to achieve a high degree of interactivity and the desire to develop a learning environment with general purpose tools and with relatively little effort, exclude each other. Figure 4 shows the tradeoff between interactivity

and cost of development. The lower levels of interactivity can be reached with domain-independent general purpose tools, but the higher levels can generally only be reached with a development effort tailored to a specific domain.

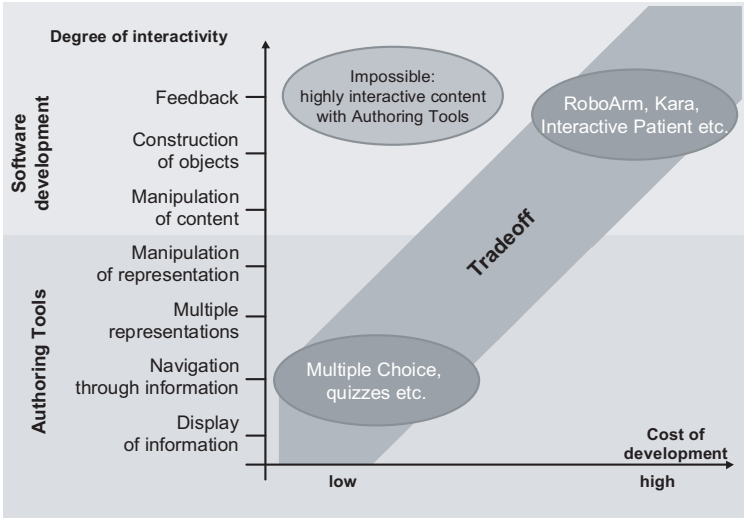

Figure 4: Tradeoff interactivity – cost of development

If we are to move from the "interpassivity" typical of today's E-Learning environments to real interactivity, we should emphasize the development of interactive, pedagogically well-designed, and domain-specific educational software. More interaction implies domain-dependent development which means it is expensive, both in terms of effort and cost. Learning is never for free, not even in E-Learning!

References

Berg, G. A. (2002). The Big Questions. *International Journal on E-Learning* 1(2), 5–6.

Bergin, R. A. and Fors, U. G. H. (2003, May). Interactive Simulated Patient – an advanced tool for student-activated learning in medicine and healthcare. *Computers & Education* 40(4), 361 – 376.

Bloom, B. (1956). *Taxonomy of Educational Objectives*. Longmans, London.

Bruner, J. S. (1960). *The Process of Education*. Harvard University Press.

Guzdial, M. and Soloway, E. (2002). Teaching the Nintendo Generation to Program. *Communications of the ACM* 45(4), 17–21.

Hartmann, W., Nievergelt, J., and Reichert, R. (2001, September). Kara, finite state machines, and the case for programming as part of general education. *Symposia on Human-Centric Computing Languages and Environments*, pp. 135–141. IEEE. http://www.educeth.ch/karatojava/.

Laurel, B. (1993). *Computers as Theatre* (second ed.). Addison-Wesley Publishing.

Mayer, R. (2001). *Multimedia Learning*. Cambridge University Press.

Schulmeister, R. (2003). Taxonomy of Multimedia Component Interactivity. A Contribution to the Current Metadata Debate. *Studies in Communication Sciences. Studi di scienze della communicazione.* 3(1), 61–80. See also http://www.izhd.uni-hamburg.de/pdfs/Interactivity.pdf

Schwill, A. (1994). Fundamental Ideas of Computer Science. *EATCS-Bulletin* 53, 274–295. See also http://ddi.cs.uni-potsdam.de/Forschung/Schriften/EATCS.pdf