Rapid Prototyping for a Virtual Medical Campus Interface

Andreas Holzinger, *Institute of Medical Informatics, Statistics, and Documentation, Graz University*

The Virtual Medical Campus Graz (http://vmc.uni-graz.at) uses e-learning to make the human medicine curriculum digitally accessible to a target population of 4,500 students and teachers. This project occurred parallel to the development of a new curriculum and thus involved all medical teachers and students. The VMC system architecture includes a multimedia repository for reusable learning objects; middleware containing the VMC logic that arranges learning objects into lectures, themes, and modules; and the user interface. The project started on 4 April 2002. Although it will run for two years, the development team's first goal was to implement a highly usable front end, presenting basic system functionalities when the new curriculum launched on 1 October 2002.

The user interface presented the main design challenge because it must support the system's acceptance and be accessible to everyone. Involving the users from the beginning and focusing on usability, not technology, addressed this challenge. In developing the VMC interface, we sought to provide only the most important and essential features, but with full commitment to good usability. To ensure the interface suited the target population, we used the user-centered design (UCD) method[1,2] (see the sidebar) and followed the general recommendations of Constantine Stephanidis and his colleagues.[3] A previous example of a successful implementation using this method appears elsewhere.[4]

Such a software project must address didactic as well as usability problems, and we consider multimedia to be one of many elements of the solution.[5] Essential curriculum changes included increased student contact with patients during the first year, bedside teaching in small groups, problem-based learning, and fewer lec-

Working under a strict timeline, these developers used simple, rapid, cost-effective prototyping techniques to create a user interface and release a working system within six months. Involving users early in the interface design facilitated acceptance.

0740-7459/04/$20.00 © 2004 IEEE

tures. Also, the curriculum is no longer divided into single disciplines but into new interdisciplinary modules, for which books aren't available. This makes the VMC an essential resource for both students and teachers, and also mandates that it present the various disciplines' content in a logical, clinically oriented context to create an integrated curriculum.

Time constraints

The development team viewed the interface development as a closed subproject of VMC system development. We had just 25 weeks to implement a fully functional front end following UCD principles (Figure 1). This extreme time pressure refined our guiding principle: Don't provide everything to a few users; just provide everyone with the necessities.

This goal emerged from our discovery during the field study that our system would be the only access point for students to gather information about the new curriculum's contents. We also realized that the teachers would only accept our system if we made it available before their respective lectures. Nobody would use a lagging system, so we focused on keeping pace with the new curriculum development.

Field research and conceptual modeling

We involved the end users from the start to ensure a system they would find easy and pleasant to operate with minimal learning effort. We spent weeks 1 through 6 of the project assembling the development team and conducting field research among the target population to define user roles, determine effective navigation methods, and gain both low-level logical and high-level functional system views.

User-Centered Design

Good user interface design is essential to ensure acceptance of new software. It is also a complex undertaking, however, encompassing many tasks undertaken in a partly parallel, partly serial, partly iterative sequence that includes planning, research, analyzing, designing, implementing, evaluating, documenting, training, and recycling or replacing. Aaron Marcus emphasizes that, like software development, user interface design usually focuses on the synthesis stages, and user interface components include metaphors, mental models, navigation, interaction, and appearance.[1]

User-centered design (UCD) methods,[2] which evolved in the human-computer interaction (HCI) field, include understanding the users and analyzing their tasks, setting measurable goals, and involving users from the project's beginning.[3] Prototyping offers a quick way to incorporate user feedback into a design. Paper-based prototyping, which relies on simple tools like paper, scissors, and stickers, bypasses the time and effort required to create a working, coded user interface. A simple form of HTML also offers a quick way of producing first scenarios.[4]

Paper mockups needn't incorporate all the frills of technology; they just have to capture the site's functionality and convey the right information.[5] Steve McConnell emphasizes that rapid prototyping doesn't simply mean coding 36 hours at a stretch, but rather involves a combination of CASE tools, user involvement, and tight timeboxes.[6]

References

1. A. Marcus, "Dare We Define User-Interface Design?" *Interactions*, vol. 9, no. 5, Sept. 2002, pp. 19–24.
2. D.A. Norman and S. Draper, *User-Centered System Design*, Lawrence Erlbaum Assoc., 1986.
3. R.J. Torres, *User Interface Design and Development*, Prentice Hall, 2002.
4. M. Rettig, "Prototyping for Tiny Fingers," *Comm. ACM*, vol. 37, no. 4, Apr. 1994, pp. 21–27.
5. D. Hix and H.R. Hartson, *Developing User Interfaces: Ensuring Usability through Product and Process*, Wiley, 1993.
6. S.C. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996.

The development team included

- The technical director, responsible for the software development

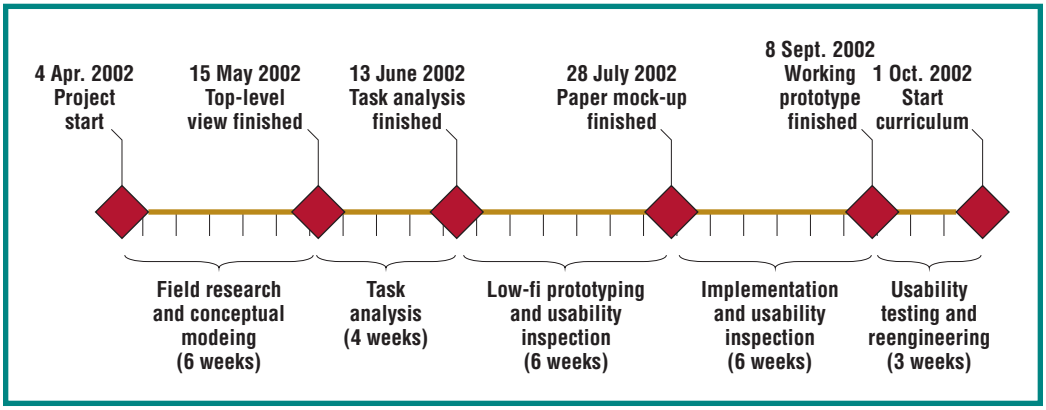


Figure 1. From conceptual modeling to a fully functional system in only 25 weeks.

We had little time to exhaustively test every workflow aspect, so we focused on what users truly needed to use our software successfully.

- The content director, responsible for the new medicine curriculum
- The Dean of Students, responsible for all official teaching affairs
- A System Analyst I, responsible for modeling and design
- A System Analyst II, responsible for modeling and design as well as some programming
- A software developer who specialized in Microsoft .NET programming
- A secretary, who handled paperwork, meeting schedules, and other coordination tasks

VMC user roles

We divided system end users into two main groups on the basis of access levels and privileges: teachers and administrators with read-write access and students with read-only access. Next, we categorized teachers by type and analyzed each type's intended workflow. We had little time to exhaustively test every workflow aspect, so we focused on what users truly needed to use our software successfully—for example, the navigational structures.

Navigational structures

Navigation, an essential interface element, addresses such important user questions as "Where am I? Where do I want to go? Am I on the right path?" To address these in the VMC system interface, the development team provided the following navigational structures from different views to access learning objects:

- By modules' training aim catalogs (for example, the human cell), an essential view for students
- By traditional or classical medical subjects (such as anatomy)
- By direct search for learning objects
- By statistics (access, navigational, or content statistics)

A catalog serves as a content directory for each module, which the new curriculum structures by topic (such as the brain or the human cell) rather than by traditional subjects such as anatomy and physiology. Every topic includes information from the various classical medical subjects.

Logical structure

The system's lowest level consists of *learning objects* grouped into lecture blocks called *lecture units*. The curriculum uses these lecture units as atomic units; each lasts 45 minutes (the duration of a university lecture) and can be a theoretical lesson or a practical exercise. The system groups these hours in *thematic blocks*, and the themes each form *modules*. The whole curriculum encompasses 32 modules taught over five study years.

Top-level view

To gain an overview of the system's main functions, we created a top-level view showing the module catalog, subject catalog, learning units, self-evaluation system, and user management and their dependencies. Based on user interviews, we developed a Mind Map (using MindManager software) in which cards (both electronic and paper) described each entity separately.

Task analysis

During the next phase, weeks 7 to 11, we analyzed the most common tasks our software would need to provide. To do so, we recruited UCD participants representing the three main user groups—administrators, teachers, and students—in order to understand the different tasks each needed to accomplish with the VMC.

Recruiting UCD participants

We recruited a group of 49 people who provided a well-balanced sample of the target population. Candidates filled out a participant profile that determined their role in the VMC, computer experience, Internet experience, and general e-learning systems familiarity. We also gathered some statistical data (age, level of education), personal comments (open-ended), and contact details, and determined the best time for UCD activities. Our final participant pool represented each end-user type, including

- Three module administrators, responsible for arranging themes and lecture hours in their modules (the teachers, as authors, can only make suggestions)
- Five subject administrators responsible for their areas of expertise (such as anatomy, physics, or chemistry)
- Twenty-six teachers, who contributed content by creating learning objects
- Fifteen students, including five beginners,

seven intermediates, and three advanced students, representing consumers of the learning objects contained in lecture hours and arranged in themes

These roles, defined within the curriculum, didn't correlate with computing experience, which ranged within groups from absolute beginner to expert.

Defining and representing tasks

The task analysis phase focused on the administrators, most of whom helped develop the curriculum and had distinct ideas about what the system needed to accomplish on a didactic level. New system development following HCI methods requires gathering contextual information, so we used interview techniques and discussions to understand and specify the context of system use, including the organizational and physical environment, user characteristics, and their predominant tasks and workflows.

Most HCI interaction models are task based, with a *task* defined as how users attain a goal. The models must take into account user competence, knowledge, and constraints. A *task representation* formally abstracts some aspect of the task, such as goals and methods, knowledge structures, or semantic components. Task limitations have also emerged as an analysis unit since HCI research moved towards the study of computer-supported cooperative work.[6]

We provided the administrator, teacher, and student groups an overview of UCD fundamentals, their specific user roles and tasks, and the project schedule. Our first sessions produced about 40 activity models (see Figure 2), one for each task from the module administrator, subject administrator, teacher, and student viewpoints. We used these models to kick off task-focused discussions with teachers and students, who, even at this early stage, contributed quite a bit.

Students, for example, said they would appreciate a thesaurus to explain difficult medical terms. Teachers raised a different concern: system security and access restrictions. We continuously discussed such issues with our software developer to determine what we could or couldn't realize within our time limit. For example, we postponed security and access restrictions—although important issues—to meet our development schedule and have a fully functional system by the start of the new curriculum.

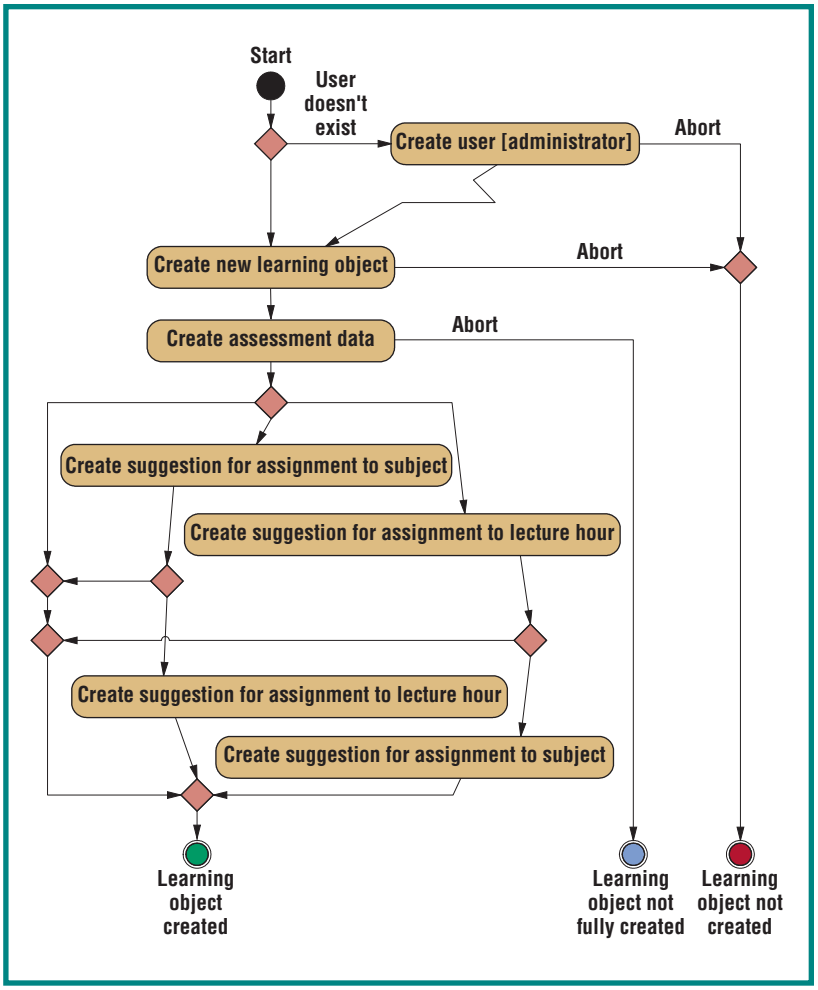


Figure 2. A typical task of a teacher authoring a learning object.

Developer involvement

The technical director, content director, and two analysts attended all UCD sessions. The software developer also attended most sessions and thus gained important insight into the large medical faculty's study life. This proved to be important because the programmer was enthusiastic about developing "his" system but was unaware, for example, that most faculty members lacked computer literacy. Involving the programmer helped us convince him that we had to build a system that even the least skilled user could easily understand and use. This ultimately made it possible to maintain our strict timeline by avoiding after-the-fact discussions concerning ease of use and possibly even a redesign of a finished or near-finished user interface.

Low-fidelity prototyping

During weeks 12 through 18, we began transforming our theoretical and discussion-

Paper prototyping and usability testing let the development team manage their risks by focusing on them earlier in the project.

based ideas into system interface elements and workflows. We started with paper mock-ups because they let us easily create experimental interfaces and modify them based on developer and user input.

Paper mock-up

Using common office supplies (paper, markers, index cards, scissors, and transparency film), we sketched the main screen and each changeable interactive interface element, including dialog boxes, menus, and error messages. The paper mock-up with its handwritten text, crooked lines, and last-minute corrections wasn't very neat but was good enough to show UCD participants and developers what the screens might look like. The mock-up provided a good basis for playing out some workflows.

One developer played the "computer" role, simulating the software's behavior by manipulating the pieces of paper. We spent three weeks doing usability tests with this paper prototype. In each test session, conducted either in the institute's usability lab or in users' offices, we asked a user from our UCD pool to perform realistic tasks with the prototype—for example, "You are a teacher; set up a theme and create some hours in the catalogue." We didn't explain or demonstrate how to use the interface prototype. We also didn't ask users for their opinions on the interface, but rather observed them actually work with it. This proved vital because some people say they love a product even if they can't use it, or can handle it well but say they hate it.

After each UCD session, the team discussed what they had seen and modified the paper prototype accordingly. We found that the developers had concerns about the interface that the users didn't even notice, and the users raised some issues we hadn't anticipated. Some changes were as simple as using a different word, moving a button slightly, or placing information in a different screen location. Other, more substantial changes included implementing a glossary. Most changes proved to be improvements in the next session.

After several sessions, we presented the final paper mock-up to the entire team, then delivered the mock-up, together with the modified specification, to the programmer. Paper prototyping and usability testing let the development team manage their risks by focusing on them earlier in the project, while they still had time to make changes.

Experimental design

For all our experiments we used equipment including

- a Hi8 video camera with a recorder, cabling, and a tripod
- a mirror
- a video monitor, so all observers could watch the camera image
- a good microphone, fixed on a tripod and acoustically decoupled from the camera to avoid interference
- headphones, because monitoring the sound proved to be essential

Our team took the following test roles:

- The technical director acted as test facilitator—administrator, moderator, manager, monitor—and handled all interaction with test users including introduction, testing, and debriefing.
- The secretary acted as data logger or scripter, recording activities, events of interest, and time of occurrence on paper (we found it essential to assign shorthand codes to expected activities before testing).
- The video operator (Analyst I) recorded all test proceedings, from initial instructions to debriefing; checked camera angles to ensure both user and interface were clearly visible; ensured proper audio recording level; and labeled, copied, and edited tapes.
- The computer operator (Analyst II) simulated computer reactions to user input using the paper prototype—for example, moving paper modules to restart the computer after system crash or provide unexpected hang-ups—and handled other computer issues.

Thinking aloud

We asked test users to verbalize their thoughts while performing tasks—what they were trying to do, questions that arose in their minds, things they found confusing, and the decisions they made. This provided abundant data with relatively few (three to six) test users.[7]

We found it necessary to prepare users during the briefing, best achieved by showing a short video clip of a previous thinking-aloud

test session. Using neutral, unbiased prompts including, "Can you say more?" or "Please say what you are doing now," or "I can't hear what you are saying," we encouraged users to keep up the information flow. We avoided directing user comments with specific questions such as, "What are you thinking right now?" "Why did you do that?" or "What are you trying to decide between?"

Cognitive walkthrough

A task-oriented interface walkthrough, analogous to a structured walkthrough in software engineering, envisions novice users' thoughts and actions and thus focuses explicitly on learnability. The interface could be a mock-up or a working prototype. This method employs the cognitive model of human exploratory learning[8]—essentially, users generally prefer to learn a new system by trial and error rather than read manuals or attend courses.[9] The exploratory learning model describes learning behavior in terms of three components:[10]

- *Problem solving:* The user chooses among alternative actions on the basis of similarity between an action's expected consequences and the current goal. After executing a selected action, the user evaluates the system response and decides whether progress is being made toward the goal. A mismatch results in an undo.
- *Learning:* When the above evaluation process leads to a positive decision, the user typically stores the action taken in long-term memory.
- *Execution:* The user attempts to find an applicable rule matching the current context. If none is found, the user invokes the problem-solving component.

Cognitive walkthrough preparation includes four steps:

- Identifying the user population
- Defining a suite of representative tasks
- Describing or implementing a paper mock-up or interface prototype
- Specifying the correct action sequences for each task

Thinking aloud, an intrusive test method, requires relatively low expertise and is used mostly during the design stage, whereas cognitive walkthrough, a nonintrusive inspection method, needs more expertise but can be used at every stage. We found cognitive walkthrough adequate for task-oriented problems. It helped us define user goals and assumptions, but it was time consuming, which prevented us from carrying out as many sessions as we had originally planned.

After we had accomplished five sessions (with one module administrator, two teachers, and two students), the development team watched the videos carefully and discussed the issues and problems raised. At the end of this stage, we delivered a final paper mock-up of the user interface (Figure 3) to the programmer so he could start building the first implemented prototype.

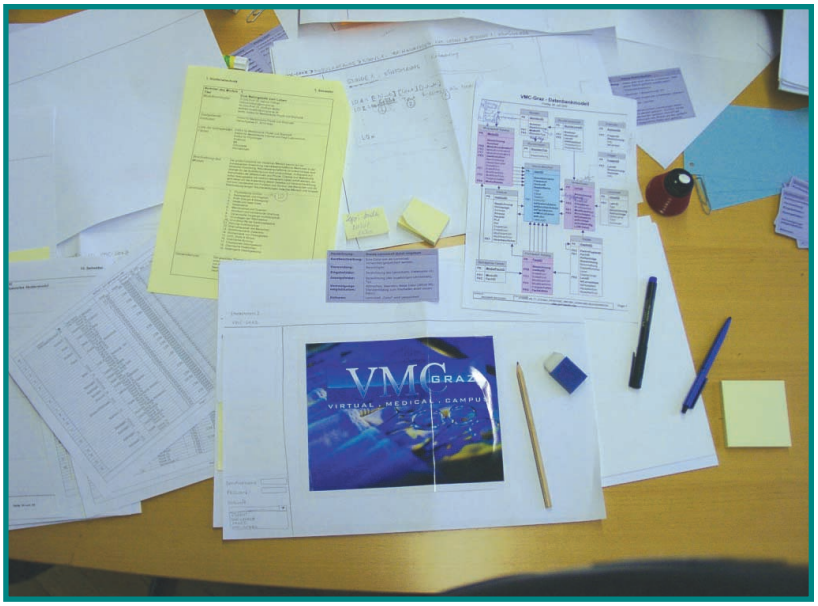


Figure 3. Module catalogue from the final version of the paper mock-up.

Implementation

From weeks 18 to 22, the programmer used the paper mock-up to create a working prototype. We chose Microsoft's .NET due to the platform's availability among faculty. This proved a good choice because although it is new and complex, it let our programmer relatively easily program gadgets that would have formerly caused developers to gape in disbelief. We could thus quickly implement functionalities that in previous projects had to be individually programmed to achieve the same result.

Of course, like all programming languages and workflows, .NET has a few internal flaws, but these are easily handled using help files and Microsoft Web pages. A disadvantage is that few people currently program with .NET, at least in Germany, making research time-consuming (although .NET is well documented).

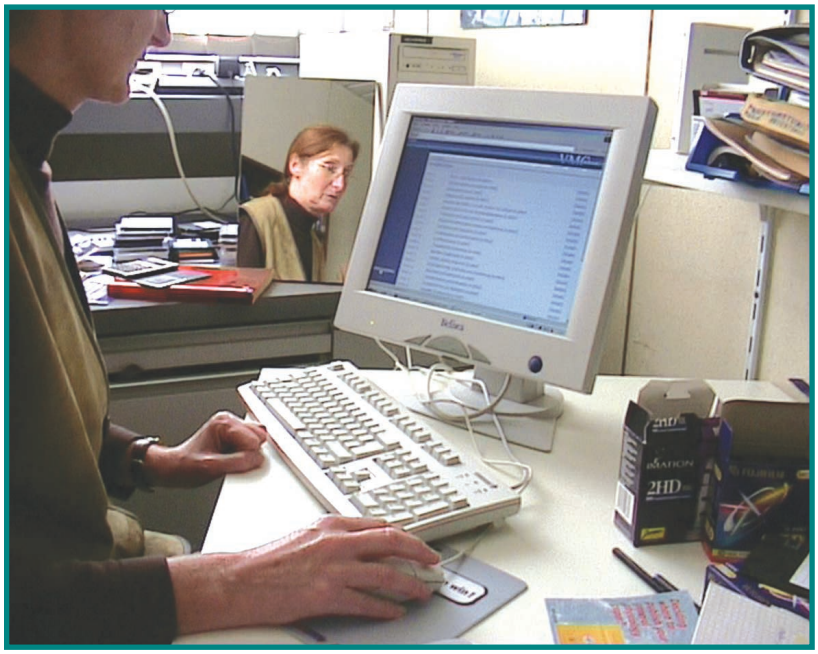


Figure 4. Twelve weeks after participating in the paper mock-up design, this teacher could now try out the same tasks online.

Usability testing and reengineering

We conducted further usability inspection with the final prototype during weeks 23 through 25. At this stage we could concentrate on specific details including fonts, font sizes, colors, entry field sizes, and so on. After thoroughly reengineering it, we put the final prototype into operation, and the real-time usability testing began. We set up our camera, microphone, and mirror in participating teachers' offices and instructed them to perform specific tasks using the prototype (see Figure 4). After six sessions, the project team (including different teachers) watched and discussed the videos. Again, we found it interesting to see how some things obvious to the programmer proved difficult for users.

Prototyping methods

The goal in prototyping is to evaluate the interface's function and flow, not its look. HTML prototypes, for example, create a polished look but don't change how quickly and easily users can accomplish a task. The paper prototype proved successful because users felt more comfortable critiquing a paper prototype than a polished coded prototype. When something doesn't work well in a beautiful interface, users will more likely blame themselves or their lack of experience than the prototype.

Paper prototyping advantages and disadvantages

Paper mock-ups worked well to demonstrate our concept and certainly proved the easiest and most efficient method. They cost little to produce and lend themselves to an easy creation of alternatives. They also encourage more suggestions because they are obviously quite changeable. Every first prototype interface has flaws, but a paper mock-up makes it easy to scribble out an element change and see if that fixes the problem. We actually saved the flawed version and explanation in case someone was tempted to try it again. We saw that both developers and users were more likely to change the paper mock-up than our (later) electronic prototype.

Interestingly, the paper prototypes weren't as easy to use as the literature described—we spent much time constructing, presenting, and experimentally analyzing them. Also, the paper versions seemed to lack face validity for the users, who sometimes didn't take them seriously enough. Our programmers in particular weren't convinced at first and proposed programming a prototype instantly. After watching the first video showing typical user problems, however, programmers also began to favor this method.

Hi-fi prototyping advantages and disadvantages

This method's biggest advantage was that users could work with the prototype directly. Some were excited to try out tasks on the real thing instead of just on paper (see Figure 4). A coded prototype already looks like the final product and provides true interactivity, a motivating factor for users.

Obviously, because these prototypes take more time to build than paper ones, alterations also took more time. We always needed the programmer's expertise to make changes, and because time is money, this method proved more expensive. Most important, some UCD participants had little computing experience and thus feared computers in general, which certainly wasn't the case with the paper mock-up.

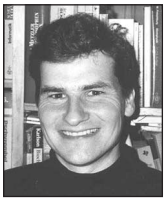
The VMC system has been online since 1 October 2002 and is proving successful in further usability studies. We attribute this success to the speedily developed interface, which the development team couldn't have accomplished without the

rapid prototyping afforded by the paper mock-ups.

Acknowledgments

I thank the VMC Graz team for their excellent work on this project, and also thank the *IEEE Software* reviewers whose substantial comments greatly enhanced the article.

About the Author



Andreas Holzinger is the technical director of the Virtual Medical Campus Graz and lecturer in information systems with emphasis on human-computer interaction at Graz University. His research interests include user-centered design, multimedia content management, and e-learning. He received his PhD in cognitive science from Graz University. He is a member of the ACM, IEEE, AACE, German Society for Informatics, German Society for Psychology, and German Society for Media in Science, and is a board member of the Austrian Computer Society. He serves as a national expert in the e-Europe e-Learning initiative. Contact him at the Institute of Medical Informatics, Statistics, and Documentation, Graz University, Auenbruggerplatz, A-8036 Graz, Austria; andreas.holzinger@uni-graz.at; www.basiswissen-multimedia.at.

References

1. *Designing Interaction: Psychology at the Human-Computer Interface*, J.M. Carroll, ed., Cambridge Univ. Press, 1991.
2. D.A. Norman and S. Draper, *User-Centered System Design*, Erlbaum, 1986.
3. C. Stephanidis et al., "Toward an Information Society for All: HCI Challenges and R&D Recommendations," *Int'l J. Human-Computer Interaction*, vol. 11, no. 1, 1999, pp. 1–28.
4. A. Holzinger, *User-Centered Interface Design for Disabled and Elderly People: First Experiences with Designing a Patient Communication System (PACOSY)*, LNCS 2398, K. Miesenberger, J. Klaus, and W. Zagler, eds., Springer-Verlag, 2002, pp. 34–41.
5. A. Holzinger, *Multimedia Basics, Volume 2: Learning*, Cognitive Fundamentals of Multimedia Information Systems, Laxmi, 2002 (also in German).
6. P.C. Wright, R.E. Fields, and M.D. Harrison, "Analyzing Human-Computer Interaction as Distributed Cognition: The Resources Model," *Human-Computer Interaction*, vol. 15, no. 1, 2000, pp. 1–41.
7. J. Rubin, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, Wiley, 1994.
8. C. Lewis et al., "Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 90), ACM Press, 1990, pp. 235–242.
9. J.M. Carroll, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, MIT Press, 1987.
10. J. Rieman, R.M. Young, and A. Howes, "A Dual-Space Model of Iteratively Deepening Exploratory Learning," *Int'l J Human-Computer Studies*, vol. 44, no. 6, June 1996, pp. 743–775.