Teaching User Interface Design and Programming to Computer Science majors

Judy Kay
Bob Kummerfeld

Department of Computer Science
University of Sydney
{judy,bob}@cs.su.oz.au

ABSTRACT

This paper describes an innovative approach to teaching about human-computer interaction to a large class of students in their third year of a Computer Science major. Because of the tight time constraints on the course and the student's learning preferences and orientation, we have created a course which interweaves the learning of several programming tools for interface construction with the development of knowledge and skills in the design of user interfaces.

We present the structure of the course, especially its most unusual aspects. We also describe our rationale for the course design, our evaluation of student learning and the response to the course.

KEYWORDS HCI education, interface design, interface programming

1. Introduction

It is becoming increasingly important that students majoring in Computer Science should learn about creating good user interfaces. At the same time, typical Computer Science programmes are very full, with little time to add courses that teach students to build better

interfaces. Indeed, as observed elsewhere (Greenberg 1996), traditional Computer Science degrees have rarely included courses in human-computer interaction.

We have designed an innovative one-semester human-computer interaction course. We needed to win over two sets of critics. First

we had to convince our colleagues that this course should earn a place in a very tightly constrained computer science curriculum. With that battle won, we knew we would need to convince our students that they should take the course over other hard-core computer science options. We also wanted the course to mesh well with the student expectations and interests since that ensures better learning outcomes. Moreover, it had to be manageable with an enrolment of 80-120 students.

Essentially, our approach was to focus on three main elements:

- interface design - the core of most courses and texts in human-computer interaction;
- interface programming - several tools in current use for interface prototyping and production;
- reflection - about the relationship between these and the students' evolving understanding of interface design.

We knew that the students would value the skills acquired in the second, programming tool, part of the course. This would attract them to the course and would motivate them. At the same time, it could serve as a driving force for learning the interface design aspects. It would provide a context for learning activities that would make the interface design issues meaningful.

This is an unconventional approach. But it avoids several potential pitfalls of a course that deals exclusively with interface design. First, and most important, students may have elected to simply not do a typical course, staying with options closer to the usual technology-centred model of Computer Science. This would have meant large numbers of students continuing to graduate without learning anything about human-computer interaction. A second risk is that many students would find a pure human-computer interaction course so different from their previous studies that they would have difficulty seeing its relevance and importance for them.

This paper describes the course from two points of view: that of the student and our own design rationale. Then we outline how our course relates to more typical courses. We conclude with a summary of the student response to the course and our experience of teaching it.

2. The student's point of view

This section presents three central elements of the student's view of the course. First we describe the practical work since that is where the deep learning occurs. With this we describe the assessment since that is a major concern for students and we have been very careful to make it serve our goals. Finally, the sequencing of lecture material and practical work constitutes the last highly visible aspect of the course.

2.1 Practical work and assessment

All practical work was based upon a single problem: creation of an interactive interface to enable computer science researchers to search and update a database of bibliographic references. The major task of the semester required students to work in groups, usually of about five. Assessment was based on a group report and demonstration.

Students built prototypes using each of the following tools: World Wide Web's html with cgi interface, tk/Python (Ousterhout 1993, Rossum 1991) an X-toolkit (Patrick 1996) and Visual Basic. In addition students implemented a small search language using yacc and lex (Kernighan 1984).

Between each of these tools, students had lectures and practical work on various aspects of HCI and how it can inform better design of interactive systems. The primary text was (Newman 1995). Practical work was done in closed lab classes with approximately seven of these devoted to learning new technical tools for constructing interfaces, four to interface design aspects and the remaining two to practice and final demonstrations, activities the last being

activities that encourage reflection and self-assessment of the work done. The four prototype implementations were done individually but with collaboration encouraged. These were assessed on a binary scale: satisfactory or not.

The final aspect of the work required students to write examination questions about the human-computer interaction lectures of the course. Each student was randomly assigned one week of lectures and required to write open-book exam questions, model answers and assessment schemes for these. Students were told that this would help them reflect on the lectures, deciding what was important and how one would assess knowing it. The best of these were made available to the class at the end of the semester. The students were told that the exam question also gave us a good idea of major problems or misconceptions. These aspects were reviewed in the last lecture.

The marks were awarded as follows:

Description	Marks
	%
Group Marks	
1. Demonstration	20
2. Report on system	10
Individual Marks	
3. Exam questions	10
4. Programming tools	10
5. Final exam	50

The report required a detailed explanation of the design of one screen in the system, an assessment of tools considered with reasons for selecting or rejecting each, evaluation of the interface created, its strengths and weaknesses and use of literature. The demonstration had to demonstrate the interface and explain its design and development process.

Note that the two main parts of the group assessment had a draft or trial run before the final one was submitted. This was to help students improve their work and also provide a low-risk

opportunity to experiment with unusual ideas or one they may have been unsure about.

2.2 Course structure

The lecture and practical work schedule interwove the two elements of the course so that students could apply new interface design skills to the new prototypes they constructed. As one might expect, students were tempted to devote practical classes to developing their prototypes. This was avoided by making the interface design tasks serve the primary goal of the semester, the interface for the bibliographic database (and the report and demonstration for it). These tasks served a dual role in consolidating lecture material and as well as aiding in team-building and group communication and management.

The actual lecture topics and practical sessions are summarised below:

Lecture Topics	Practical
1. Overview of course, www/html	
2. forms/cgi	html, cgi
3. screen design	html, cgi
4. Python/tk	screen design
5. usability	Python/tk
6. usability	usability
7. X toolkit	usability
8. user-centred design	X toolkit
9. Visual Basic	design
10. formal notations	
11. yacc/lex	Visual Basic
12. interface problems, Java	draft reports
13. cognition, mental models	
theoretical foundations	practice demos
14. conceptual design, review	final demos

3. Design rationale - the teacher's view

There were many elements to our design. Here, we focus on two of the most important. In terms of learning effectiveness, the major aspect is the students' practical experiences. So we

explain their rationale first. The other quite unusual aspect is our ordering of the teaching and learning elements of the course. Both these are inspired by a problem-based approach to learning where the demands of a problem are used to motivate the learning.

3.1 The role and management of the practical work

The course is dominated by the single major practical problem. Since the students had to make individual attempts at the prototypes they built with each of the tools, they had the opportunity to rebuild the same system four times before creating their final work in the interface as part of the group effort. This had several advantages:

- with each new prototype, they had a growing knowledge of interface design to draw upon;
- and they had a growing body of experiences both of building the earlier interfaces and seeing the efforts of their peers;
- there was no additional start up cost for getting into each new programming task as they were rebuilding interfaces for the same tasks and, at the same time, working towards the semester's main problem;
- the comparison between the tools were meaningful in two senses: they were doing the same tasks; and they were part of working towards a goal, the group's final project;
- reflection about what had been learnt was a natural part of the process.

Overall, we believe that the whole practical and learning sequence constitute an 'authentic' problem for real world interface design, involving just the stages practiced in the course: prototyping to explore interface tools and designs, learning about HCI that can improve the design and self-assessment at each stage to inform future work.

The role of groups in the major submissions is important. Since the students had built prototypes individually, they had to come to terms with the differences in their individual approaches when they came to create the group's

interface design. Each group had to reach agreement on the tool they would use. The decision balanced ease of programming against power for interface construction. We encouraged each practical class to try diverse tools and to look at each other's work. In the end, the bulk of implementations were split between use of www-html with various languages in the cgi-scripts and the X-toolkit.

The group was critical to effective learning about the range of usability and design approaches. Group work helped students reflect about their work and reasoning since they needed to discuss it with group members and justify it.

It also allowed work to be planned by the group, with a sharing of the effort required to do the actual construction, evaluative and research tasks and the whole group would assess and review outcomes.

Another critical aspect of our course management is that we took great care to ensure that students should not be overloaded. The course load was spread right through the semester and students were encouraged to collaborate at all stages of the programming work. We wanted students to have the time to stand back from their work and reflect on the learning experiences. To permit that, we had to make sure there was enough time for students to reflect as well as do the required tasks in the curriculum.

The overall effect of the practical work design was that the final products were extremely impressive. The demonstrations were very good, enhanced considerably by the practice sessions and the reports also demonstrated that students learnt from feedback on their draft submission. The class atmosphere was positive and cheerful as students worked co-operatively and purposefully towards the semester's goals.

3.2 The learning sequence

Two principles underly our ordering of the course elements:

- the programming and design elements were to

be tightly linked and interwoven so presentation alternated between them;

- aspects that students were likely to relate to were presented early in the course and these formed a basis for the others that were treated later.

Given these principles, it is easy to see that we would start with a programming tool. It represents the sort of learning our students are familiar with and value. Moreover, the choice of `www/html/cgi` programming enabled students to devote time learning about an area that is currently of great interest to them, employers and the public.

At the same time, it makes a very natural route for exploration of some truly ghastly screen designs, the first part of the human-computer interaction side of the course. But this is not the primary reason for teaching that very early. Screen design has several aspects that make it perfect for introducing a user interface design course. Firstly, there are many, well-defined facts one can learn. Applying these will reduce some of the most terrible screen design errors. This aspect can be taught in terms of very concrete guidelines with solid theoretical, empirical or physiological bases. For example, use of saturated blue text is demonstrably unreadable and we can explain why. Essentially, screen design makes a *concrete* element which has high payoff for a little learning. It makes for a rewarding start to the course.

While the practical classes consolidated the lectures on screen design, lectures introduced Python/tk. This was primarily to introduce an example of a scripting language and the language, Python, which was also useful for cgi-scripts.

The choice of usability as the second main interface design topic was driven by its immediate relevance and applicability. We linked this to the familiar software engineering concern with testing and made it clear that this was a new collection of strategies to build testing into the

design process as well as the final evaluation. Students responded well. Their application of these aspects was visible in the final products (especially when compared with the prototypes). The reports of usability and the excellent exam questions submitted indicated a good level of understanding.

The next programming tool was GraphApp, an example of an X-toolkit designed to be easy to learn. For programmers familiar with C, it is a painless introduction to central concepts of this style of event driven interface programming.

At this point, we gave a brief overview of user-centred design and drew heavily on the student's background in software engineering as a basis both a foundation and as a basis for comparison. With the time available, our primary concern in this topic was that students have some experience and awareness of approaches they should draw upon in later work. We consider this satisfactory since we feel that the next significant level of learning in this area would require far more time than we had. Our introduction gave a useful foundation.

The next tool, Visual Basic, was chosen because it is so widespread that we felt our students would value meeting it. It was very easy to make simple prototypes and represents a style of interface that has an important place.

From this point we began the aspects of HCI that were less immediately useful for the project but which were useful for the report and demonstrations. We introduced various formal notations, as communication tools during the design process and on completion. Students seemed to like object-state transition diagrams and the User-Action Notation and other formal approaches: most of the exam questions written for these lectures focused on the most formal aspects. We linked this to our very brief survey of interaction styles. The lecture material simply summarised the interface styles that were familiar from the practical work and then introduced

those interaction styles not available in the practical work.

The final tool may seem unusual in any interface course. It introduced students to yacc and lex, tools for building parsers for small languages. By careful construction of a practical task closely linked to the main project and by providing scaffolding, students could see the potential of these tools. There are many situations where the user may benefit from the power of a small command or programming language. This part of the course aimed to ensure that when our students met these cases, they would look to such tools to facilitate building that part of the interface well.

The last three weeks of lectures were mainly devoted to theoretical and psychological aspects that are foundations for all the pragmatics of the earlier HCI material. So, for example, it was at this stage that we introduced the human information processing model: students were readily able to see how this related to the GOMS analyses they met earlier. (We know this because our lecture style involves stopping at suitable points in the lectures and posing questions for small group discussion then sharing with the class. This case is typical of how we used this lecture style, asking students to stop and think about how new material related to what they already knew.) This section included some of the classical and entertaining examples of interface failures. These introduced mental models, various cognitively based theories, models of task performance, theories of exploration and conceptual design methods.

Rather than detail all topics discussed, we note that the lectures took a similar perspective to the textbook (Newman 1995) and covered all of its content except the chapter on statistical bases for design and analysis of experiments. We also supplemented lectures with some material from books (Baecker 1987, Borenstein 1991, Mayhew 1992, Galitz 1993, Preece 1994) and these also served as class references. As a brief

interlude of change in this period, there was one lecture on Java.

4. Related courses

The most unusual aspect of our course is the integration of programming and design aspects of interface creation.

Another innovative aspect of the course is the ordering of the HCI material which differs from most texts, (Baecker 1987, Preece 1994, Thimbleby 1990, Dix 1993, Lewis 1996) including our prescribed text (Newman 1995) as well as the excellent ACM SIGCHI curriculum materials that include example curricula (SIGCHI).

It is interesting that a study of practicing user interface designers (Howard 1993) identified three central knowledge bases that practitioners should have: the technology of the user interface, facility with interface design and, particularly for the software engineer, skill in programming the user interface. The underpinning and history as well as theoretical aspects were judged as being of less importance. Our approach should enable students to build strongest foundations in areas that Howard's study suggest are critical.

5. Evaluation

There have been several indications that students have responded well to this course. Firstly, it attracted and held the bulk of the eligible candidates. Students responded energetically to all elements of the course: lecture and practical class attendance were good.

In terms of learning outcomes, the standard of the final systems was incomparably better than those as prototypes. They reflected the usability assessments students had done and the formal material from lectures. Some showed the effects of considerable research, exploring similar classes of systems and insightful analysis of user trials. The reports indicated that students were often aware of shortcomings of their designs as well as the trade-offs they had to make

against conflicting goals. The demonstrations were also a good indication of student awareness of interface design issues. We received unsolicited positive comments from staff taking major software projects where they were struck by the dramatic rise in quality of interface design by students doing our course.

The student's examination questions, answers and grading schemes indicated a good understanding of major issues. A small number of students sat take-home final examinations and these indicate a good understanding of the main HCI aspects of the course. The final examination performance was consistent with our other source of evaluative information.

There are, as one would expect, several small areas where we would like to see improvements. For example, students should be encouraged to read more. Another serious concern to address in future is that students need more consolidation of the fact that their intuition is not enough. Some did appreciate that this was one of the central messages of the course. However, some casual student' comments indicated they only knew this at a superficial level because they made slips in conversation, belying the lack of deeper understanding.

Finally, student response to the course has been very positive. On the final course evaluation questionnaire, only two responses of the 71 received were clearly negative: these two students seemed to dislike the HCI part and did not see its relevance. Overall the responses were resoundingly positive, with many students taking the time to write quite detailed comments on what they appreciated about the course. We have also had several inquiries from students who would like to continue this work into Honours year projects.

6. Conclusions

For each of the elements of the HCI part of the course, one could easily justify far deeper study than our course permitted. Indeed, each

aspect is the subject of whole texts that are designed for use in full semester courses. We cannot claim to have given our students the depth of understanding that a half dozen HCI courses could give! However, we did aim for a useful foundation. For each area, we aimed to achieve the following:

- students met central ideas, terms, concerns and practical value of that area. This gave more than a mere literacy level of learning: it provided a foundation for further learning when problems demand it;
- students should be aware of the limits of their knowledge so that they will *know* that they need to go to the literature and learn more as problems demand serious application of elements of the course;
- students apply something of what they heard in each set of lecture topics to the main task of the semester and so built a deeper understanding strongly linked to their other knowledge;
- concrete and high leverage learning areas were treated early and more theoretical foundations were treated later when students could see them as the underpinnings of the concrete detailed knowledge they had by then applied to building better interfaces.

The common thread of the practical work served as an important way to unify and link the series of learning experiences. We are convinced that the time taken by the interface programming parts of the course provided advantages for learning the HCI aspects: the new HCI knowledge could immediately be linked to an authentic problem and the motivational and learning benefits help to offset the fact that the HCI aspects had to share time with the programming. Moreover, we believe there should not be a line between design and programming since a useless or difficult-to-use program may as well not be written at all. The pure technology has meaning and utility only when it can be combined with design knowledge. Our innovations in this course for teaching interface design and programming make it an excellent HCI basis for

students who can only fit one HCI course into their computer science major.

References

- Baecker 1987.
 Ronald Baecker and William Buxton in *Readings in human-computer interaction : a multidisciplinary approach*, Morgan Kaufmann (1987).
- Borenstein 1991.
 N S Borenstein, *Programming as if people mattered: friendly programs, software engineering, and other noble delusions*, Princeton University Press, Princeton, New Jersey (1991).
- Dix 1993.
 Alan Dix, *Human-computer interaction*, Prentice Hall (1993).
- Galitz 1993.
 Wilbert Galitz in *User-interface screen design*, QED Pub. Group (1993).
- Greenberg 1996.
 S Greenberg, "Teaching Human Computer Interaction to Programmers," Research report 96/582/02, Department of Computer Science, University of Calgary (1996).
- Howard 1993.
 S Howard, *What do user interface designers say they need to know about HCI?*, OZCHI'93, Canberra Australia (1993).
- Kernighan 1984.
 B W Kernighan and R Pike in *The Unix programming environment*, Prentice-Hall (1984).
- Lewis 1996.
 Clayton Lewis and John Rieman, *Task-Centered User Interface Design*, ftp.cs.colorado.edu in: /pub/cs/distribs/clewis/HCI-Design-Book (1996).
- Mayhew 1992.
 Deborah J. Mayhew in *Principles and guidelines in software user interface design*, Prentice Hall (1992).
- Newman 1995.
 W M Newman and M G Lamming, *Interactive System Design*, Addison-Wesley (1995).
- Ousterhout 1993.
 John K Ousterhout, *TCL and the TK Toolkit*, Addison-Wesley (1993).
- Patrick 1996.
 Lachlan Patrick, *GraphApp*, <http://www.cs.su.oz.au/~loki/graphapp/> (1996).
- Preece 1994.
 Jenny Preece, Yvonne Rogers, H Sharp, D Benyon, S Holland, and T Carey in *Human-computer interaction*, Addison-Wesley (1994).
- Rossum 1991.
 Guido van Rossum and Jelke de Boer, "Interactively Testing Remote Servers Using the Python Programming Language," *CWI Quarterly*, 4, 4 (1991).
- SIGCHI.
 ACM SIGCHI, *Curricula for Human-Computer Interaction*, (<http://www.acm.org/sigchi/cdg/cdg3.html>).
- Thimbleby 1990.
 Harold Thimbleby, *User interface design*, Addison-Wesley (1990).