Extending HCI in the Computer Science Curriculum

Andy Cockburn and **Tim Bell**
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
{andy,tim}@cosc.canterbury.ac.nz

Abstract

This paper discusses the teaching of Human-Computer Interaction (HCI) at opposite ends of the Computer Science course curriculum. We provide tips on course content within final-year HCI courses. These tips focus on interface dialogue notations which are often either ignored or superficially discussed in HCI texts. By teaching these notations through the specification of real interfaces, their value becomes clear and we ease the problems of students viewing HCI as "woolly and vague". We also describe the ways that we are introducing HCI lecture material into our first year Computer Studies service course. The benefits that we hope to gain include the promotion of students' critical insight into the software systems that they learn, and increased confidence through a reduction in the students' tendency to blame themselves for the problems encountered while using software systems.

1 Introduction

In the Computer Science department at the University of Canterbury we are finding that Human-Computer Interaction is extremely popular with students. Regardless of its popularity, we believe that the study of HCI is beneficial for students at all levels of the computing curriculum, even those taking non-advancing computer studies "service" courses. In this paper we describe some of the non-standard ways that we are integrating HCI into our computing curriculum, the benefits that have been derived (or that we expect to gain), and the techniques that we have used to increase the formality of user interface design and analysis.

Human computer interaction (HCI) is a relatively new addition to the computer science curriculum. Following Denning *et al.*'s recommendation [3], the 1991 ACM/IEEE joint curriculum included Human Computer Interaction as one of the nine "core" areas of computer science. In order to assist in the design of HCI study programmes, the ACM Specialist Interest Group on Human-Computer Interaction (ACM-SIGCHI) produced a curricula for Human-Computer Interaction [7]. Several HCI practitioners have published descriptions of their courses [5, 8, 12], and low-detail outlines of one hundred and sixty nine courses are accessible through the HCI Education Survey [11].

Although extremely valuable, these resources do not provide advice on many of the problems commonly associated with undergraduate HCI courses. These problems include text books treating the more formal methods of user-interface dialogue specification in a light-weight and unconvincing manner, and the consequence that the more technically oriented computer science students view HCI lecture material as either "obvious" or "woolly and vague".

Beyond the problems of teaching HCI, we believe that HCI education can act as

both a morale-booster and a tool for critical insight within introductory computer studies "service" courses. To our knowledge, this potential use of HCI education has been unexplored.

In section 2 we describe the techniques that we have used to overcome the common problems of HCI within Computer Science, and in section 3 we describe the motivation for introducing HCI into our large first year Computer Studies service course which teaches the fundamentals of computer use.

2 HCI within Computer Science

A common misconception held by students entering their first course on HCI is that they will be studying graphical layout, "getting the buttons in the right place" and general presentation issues. Several HCI texts do little to disabuse students of this belief, providing a highly superficial and unconvincing overview of the methodologies that make up "best practice" in HCI.

On graduation, many of our students will find jobs that include designing and coding graphical user interfaces. It is important that the new generation of computing professionals can introduce to their companies the skills needed to design, and communicate the designs of, well considered user interfaces. Competency with complex user-interface toolkits and languages (such as the Java AWT, or Xt and the Motif widget set) is, for many Computer Science students, a natural bi-product of well-honed programming skills, but the ability to use these skills to produce excellent user interfaces is best promoted through explicit instruction on HCI.

At Canterbury, the course with the largest HCI component (20 lectures) is a final year Software Engineering paper. Details of the paper, which includes a large group software development project, are provided in [2]. The recommended text for the HCI component of the course is "Human-Computer Interaction" by Dix, Finlay, Abowd and Beale [4]: selected because it provides a more formal analysis of HCI than most. Table 1 provides an outline of the topics taught in the HCI portion of the course.

Many portions of the course are similar to those of other HCI courses, such as Greenberg's [5]. The primary difference, however, is the focus on notational techniques for user-interface specification and analysis. Students are continually encouraged to use their Computer Science skills to focus on the underlying states within the interface. Excellence in interface design, it is argued, is emergent from well-considered design of the underlying interface state transitions; excellence in graphical presentation merely provides a polished facade. Notational mechanisms provide a concise and precise description of the interface that supports communication of the interface between designers, and they illustrate the designer's model of the system (Figure 1) without the impediments of the interface facade.

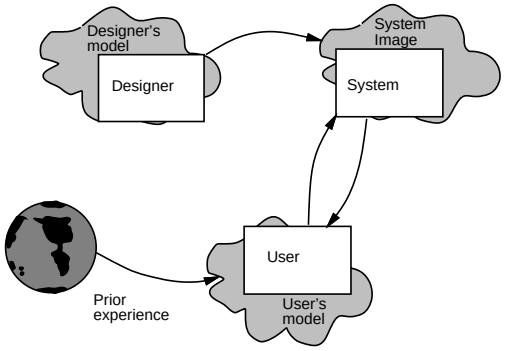


Figure 1: Norman's model of interaction and system design

Several dialogue specification and analysis techniques are taught within the course, including Backus Naur Form (terminals representing the user's input and non-terminals representing system states and modes), state-transition diagrams, statecharts, production rules and User Action Notation (UAN) [6]. All but the last of these techniques are discussed in the course text, but the coverage is superficial, and the examples are rather unconvincing. We have found that examples based on real systems are very motivating for the students, for two main reasons. First, when examining interfaces specified with the notations, curious interface properties are easily detected and pin-pointed. Without the specification, detecting and articulating the location of the errors would be much more difficult (often requiring extensive interaction with the system). Second, the easy detection of interface errors (and curious properties)

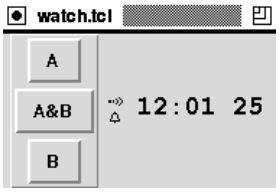
Topic	Lects	Summary
Introduction and Overview	1	Administrative details and motivation.
The human	2	The human as an information processor. Elementary psychology. Human phenomena.
The computer	1	Ergonomics. Implications of finite processor speed. Alternative input mechanisms: marking menus, unistrokes and magic lenses.
Interaction	3	Errors and mistakes. User models, system image, designer's model. Interaction frameworks. Paradigms for interaction.
Usability principles	3	Encapsulation of best practice within guidelines for design and evaluation.
User centred and task-centred design	2	Designing with the users. Think-aloud evaluations. Iterative design, storyboarding and prototyping.
Rationalised design	4	GOMS models. Dialogue notations: BNF, STNs, state-charts, production-rules, UAN.
Formal evaluation	2	Controlled experiments.
Windowing systems	1	Elements of windowing systems.

Table 1: Topics taught in the HCI segment of the Software Engineering course.

```
System-states: {TimeSet, SetSecs, SetMins, SetHours, SetNumdays,
      SetMonth,SetWorddays, Date}
Input-events: {A_du, B_du}
Output-states: {flashtime, flashsecs, flashmins, zerosecs, advmins,
      advhours, flashhours, showdate, flashnumdays, flashworddays, advmonth,
      advnumdays, flashmonth, advworddays}

A_du : TimeSet --> flashsecs SetSecs
B_du : TimeSet --> showdate Date
A_du : SetSecs --> flashmins SetMins
B_du : SetSecs --> zerosecs
A_du : SetMins --> flashhours SetHours
B_du : SetMins --> advmins
A_du : SetHours --> showdate flashnumdays SetNumdays
B_du : SetHours --> advhours
A_du : SetNumdays --> flashmonth SetMonth
B_du : SetNumdays --> advnumdays
A_du : SetMonth --> flashworddays SetWorddays
B_du : SetMonth --> advmonth
A_du : SetWorddays --> flashtime TimeSet
B_du : SetWorddays --> advworddays
```

2a. Production-rules specification (time and date setting states only).



2b. The watch simulation.

Figure 2: The Pulsar watch laboratory: specification and simulation.

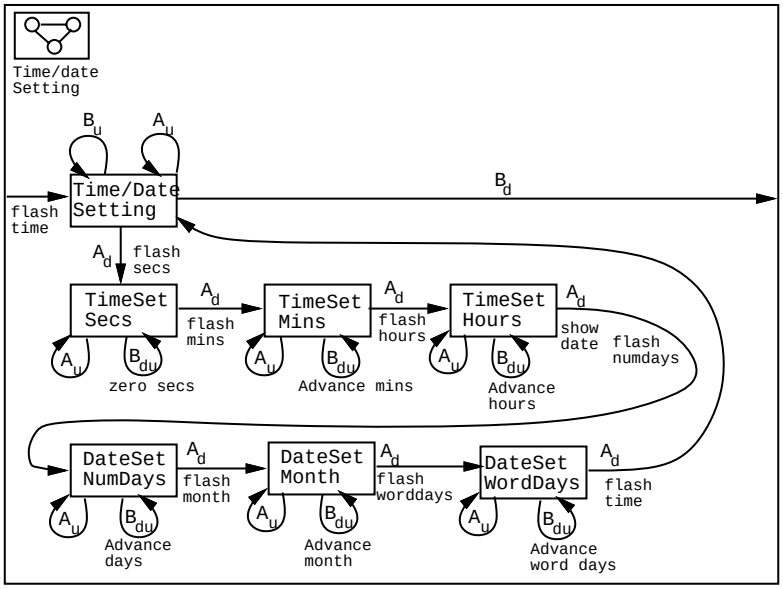


Figure 3: State transition diagram specification for time and date setting states of the Pulsar watch.

acts as a point of "topic validation" for the students: clearly the errors would not have propagated through to the finished product if these techniques had been used.

Like many topics in Computer Science, comprehension of the notational specifications is very much simpler than production. In laboratory sessions the students analyse interfaces and produce specifications in the various notations. One interface that they specify is a precise simulation (written in Tcl/Tk) of a Pulsar digital watch—a high functionality model. Figure 2b shows a screen-dump of the running simulation. The buttons `A` and `B` simulate depressions of the top and bottom watch buttons. The button labelled `A&B` simulates a simultaneous depression of both buttons. The production rules that specify the time and date setting mode within the interface are shown in Figure 2a. Figure 3 shows the state-transition diagram specifying the time and date-setting portion of the interface. The code for the simulation and the lab handout that describes the work can be accessed from `http://www.cosc.canterbury.ac.nz/~andy/314`.

It is important to demonstrate to the students the strengths and weaknesses of the various notational mechanisms. We also note the relevance of the Sapir-Whorf hypothesis [13] in this regard: that the different expressive capabilities of languages and notations influences both the way we describe interfaces and the level of description. One example of this is the difference between the level of description in the production rules watch specification (Figure 2a) and the state-transition specification (Figure 3). In the versions shown, the production rules provide less information about the interface than the state-transition diagram because user actions are condensed into "clicks" (for example, `A_du` means button `A` down and immediately up) rather than into temporally separated combinations of a {button-down, button-up} events. Consequently, the production rules specification incorrectly implies that there is no feedback to the user until after the button-up event. The state-transition diagram shows that this is not the case: rather, the new state is attained on the button-down event, and the button-up event is redundant. Naturally, the production rules specification *could* be modified to correctly specify the behaviour, but the additional burden of doing so may discourage designers from specifying such styles of interaction in the first place.

The watch example provides students with an example of a highly modal and sequential interface. In contrast, most graphical user interfaces have few modes and have many

Figure 4: The interface to `workman`.

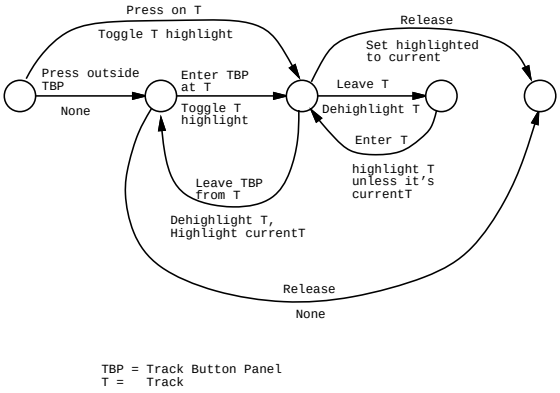


Figure 5: State transition diagram of track selection in `workman`.

concurrent portions in their dialogues. A good demonstration of this (and another good example of an interface with curious features) is provided by the graphical user interface to `workman`, a Unix-based CD player shown in Figure 4. Although `workman`'s track selection interface looks simple (and actually is simple to use), it contains some bizarre properties when the interface is fully described. Clearly, the full range of possible user interaction with the system was not specified, and many coincidental interface properties were inherited from quirks of the system's implementation.

A past student assignment involved translating a necessarily long English description of `workman`'s interface to track selection into a UAN description and into a state-transition diagram. The UAN description (Table 2) can be extremely concise. The state transition diagram (Figure 5) is rather more problematical because the interface is not highly sequential and because of the difficulty of handling the context-dependent feedback provided by the system (for example, the weak label on the arc "highlight T unless it's currentT"). The resultant complexity of the STN belies the relative simplicity of the interface: again, a potentially important lesson for the students.

3 HCI within Computer Studies Courses

The Computer Science department at Canterbury teaches a large (900 student) first year "Computer Studies" course that is designed to provide new University students with general competence in computer use. Courses such as this are problematical because University courses are supposed to distinguish themselves from Polytechnic and other courses by encouraging critical insight and by imparting underlying principles. Careful course design has partially resolved this issue by changing the focus of the practical part of the course to be task oriented rather than software oriented, focusing on general principles and using the software only as a means to an end [1]. Nevertheless many still regard the course as a "skills" course rather than an academic one. A second problem associated with fundamental computer studies courses (distinct from computer science) is that many of the students lack the confidence to learn through experimentation, and many are intimidated by the software that they are learning.

We believe that teaching rudimentary HCI within our Computer Studies will ease both of these problems. First, by teaching fundamental models and principles of Human Computer Interaction we hope to promote critical insight into the students' experiences with the software they are learning. Second, we intend that this critical insight will become a lever for the students' learning and confidence. Although many university students are confident computer users, there is a significant number who feel very nervous and intimidated. Our hypothesis is that when students encounter difficulties with their software packages, rather than blaming themselves for their own lack of understanding, they will apply more rational

TASK: Select, de-select, change track		
User Actions	**Interface Feedback**	**Interface State**
($\sim [T \ in \ Tracks] \ M^{\vee}$ $\mid \sim [x, y \ not \ in \ Tracks] \ M^{\vee}$)	$T{-}!$: $T!$ $T!$: $T{-}!$	
($[T \ in \ Tracks] \sim$ $\mid \sim [T \ in \ Tracks]$ $\mid [Tracks] \sim$ $)^*$	$T{-}!$ $T \neq currentT$: $T!$ $T = currentT$: $T{-}!$ $currentT!$	
$M^{\wedge}$		$T!$: $set \ currentT = T$

Individual track buttons denoted by T.
Entire track panel denoted by $Tracks$.

Table 2: User Action Notation Specification

analysis on the cause of their difficulty. For example, when the system carries out an unexpected action or when it fails to carry out the expected action, we would hope that students would recognise that there is a mismatch between their own mental-model of the system's operation and the designer's mental model of system operation (Figure 1). Alternatively, it could be that the designer's mental model and the user's mental model are in unison, but that system-image is poorly communicating the internal (designer's) model to the user.

We are implementing these changes in our Computer Studies course this year. The overall lecture content of the course has not been dramatically altered by the introduction of the HCI lecture material. The main difference is that whenever specific software is discussed (such as a word processor, or the operating system), the usability of the interface is viewed critically. The learning goal is to provide tools with which students can assess the relative merits of the software systems that they are learning. Nielsen's ten usability heuristics (Table 3, from [9]) are presented to the students, and then used as a benchmark by which systems are judged. Examples of usability problems abound in commercial software, and by teaching an HCI perspective, this impediment to teaching computing becomes an asset!

The students are given examples of how software follows or diverges from these guidelines, and are encouraged to think critically about the usability of the software that they are using. The following are some of the examples given for the ten heuristics.

Simple and natural dialogue Some office productivity tools confuse the user with a myriad of buttons and controls. On the Macintosh, a floppy disk must be dragged into the trash can to eject it, which is hardly a natural application of the desktop metaphor.

Speak the user's language The classic example is the instruction "Hit any key to continue"; what if the user really does *hit* a key? Or if they look for a key labelled "any"? Or if they press the shift key, which does nothing? Or if they press the escape key, which might halt the program!

Minimise the user's memory load A positive example here is the way cut and paste are used consistently amongst different programs; once the concept has been learned then it can be re-used many times.

Consistency While the behaviour of the copy and paste commands is generally predictable, on certain spreadsheets they operate quite differently, both in the way that data is selected, and how the data appears after it has been moved.

Feedback One of the first tasks that many students ever perform on a computer is

Heuristic	Description
Simple and natural dialogue	Minimising complexity, principles of graphical layout.
Speak the user's language	Affordances, mappings, metaphors and using the user's perspective.
Minimise the user's memory load	Limits of human short-term memory load. Recognition versus recall.
Consistency	Importance of generalisation in learning and use.
Feedback	Types, persistence and response times.
Clearly marked exits	Cancel, undo and action priorities.
Shortcuts	Power-user options including macros, history and agents.
Good error messages	Guidelines for error messages.
Prevent errors	Interface modes, interface syntactic correctness and commensurate effort.
Help and documentation	Task centred minimal manuals.

Table 3: Nielsen's ten usability heuristics.

to enter a password. By necessity, the computer gives little feedback in this situation, and the process can be very difficult, particularly if the caps lock key is down accidentally. Lack of feedback is also a problem if students are using a slow computer or network, and user actions are queued up and executed well after they have been performed.

Clearly marked exits An example is the Windows operating system, which has the shutdown command on the "Start" menu.

Shortcuts GUI operating systems generally allow the user to make a "shortcut" or "alias" for a file, so you can put an image of it on the desktop or in a menu for easy access. Students are also encouraged to get to know keystrokes for common actions such as the one to close a window.

Good error messages Some applications produce error messages that are very cryptic (e.g. "Error number 23") or vague (e.g. "There was a problem opening the file").

Prevent errors A simple example is that the copy and cut commands are made unavailable if nothing is selected. It is not hard to find systems that offer a command, and then say that it can't be done when it is chosen.

Help and documentation Often documentation and on-line help is poor. Students soon find this when they search for help on-line. If the system uses different terminology to the student then the student may never find help even though it is available.

In addition to viewing other work critically, the students are also able to engage in activities that require them to apply HCI design principles themselves, despite the fact that they not do any programming. For example, the way a student organises their files and folders on the desktop is a part of the user interface that is under their control (e.g. Miller's 7 ± 2 principle can be applied to the number of files in a folder). Other examples include the layout of a spreadsheet (which will have inputs, outputs, and processing sections) and a Web page (which are notorious for having confusing interfaces).

The idea of forming a suitable model for software is emphasised, which is illustrated by Figure 1 from [10]. Students are introduced to the idea that user errors are typically either slips, or result from an incorrect model, which can help them account for errors, and view their errors more as a learning opportunity than a hindrance.

An HCI perspective on other topics which are normally taught within elementary Computer Studies courses will, we believe, increase the level of 'relevance' to the students. For instance, most Computer Studies courses include lectures on hardware which relate the various components of computer systems. Our course evaluations have frequently shown that students find this material dry and irrelevant.

Within an HCI perspective, hardware components can be introduced as part of the "User, Input, System, Output" communication cycle, within a "relevant" context of, for instance, "why does this user action take a long time?"

4 Conclusions

Introducing HCI to the non-advancing computer studies course makes the course much more satisfying to teach, and encourages critical thinking from the users. Even some of the apparently simple tasks such as designing a Web page or organising files give students the opportunity to apply the principles. There may even be an opportunity for these informed users to provide more useful feedback and suggestions to industry. Unlike computer "experts," whose design may be coloured by what can be implemented, an informed user is primarily interested in what is useful.

In our final year HCI course we have found that the use of realistic examples, which go far beyond those offered in text books, greatly assist the students' appreciation of the value of notational specifications of user interfaces. Although we do not claim that notational specification of user interfaces is sufficient for the development of excellent user interfaces, we do claim that the notations provide a powerful additional tool that should be included in undergraduate HCI courses.

References

[1] T Bell. An authentic task-based university level computer literacy course. In *Proc. First Australian Computer Science Education Conference*, pages 295–301, Sydney, Australia, April 1996.

[2] N Churcher and A Cockburn. An immersion model for software engineering projects. In *ACM Australasian Computer Science Education Conference '97. Melbourne, Australia. 2–4 July.*, pages 163–169. ACM Press, 1997.

[3] PJ Denning, DE Comer, D Gries, MC Mulder, A Tucker, AJ Turner and PR Young. Report on the ACM task force on the core of computer science. ACM Press, 1988.

[4] A Dix, J Finlay, G Abowd and R Beale. *Human-Computer Interaction.* Prentice Hall, 1993.

[5] S Greenberg. Teaching human-computer interaction to programmers. *Interactions*, Volume 3, Number 4, pages 62–76, 1996.

[6] HR Hartson, AC Siochi and D Hix. The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Office Information Systems*, Volume 8, Number 3, pages 181–203, 1990.

[7] B Hefley (editor). *ACM SIGCHI Curricula for Human-Computer Interaction.* ACM Press, 1992. ISBN 0-89791-474-0.

[8] D Hix. Teaching a course in human-computer interaction. *Computer Science Education*, Volume 1, Number 3, pages 253–268, 1990.

[9] J Nielsen. *Usability Engineering.* Academic Press, 1993.

[10] DA Norman. *The Psychology of Everyday Things.* Basic Books, 1988.

[11] G Perlman and J Gassen. HCI education survey. `http://www.acm.org/sigchi/educhi/`, 1994.

[12] C Phillips and E Kemp. Towards the integration of software engineering and HCI education: A cross-disciplinary approach. In *OzCHI'96: The Sixth Australian Conference on Computer-Human Interaction.* Hamilton, New Zealand. November 24–27., pages 145–150, 1996.

[13] H Thimbleby. *User Interface Design.* ACM Press, Addison-Wesley, 1990.