Transactions on Information and Communications Technologies vol 7, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

Teaching usability to visual basic programmers: a case study

B.R. Webb

School of Finance and Information, The Queen's University of Belfast, Belfast, UK

Abstract

This paper describes the delivery and outcomes of a new postgraduate course in User Interface Design. The objective was to embed usability within software engineering instruction through iterations of evaluation and design. Visual Basic was the prototyping tool used. The experience is contrasted with that of others teaching usability and the unique contribution of Visual Basic is discussed.

Introduction

In the debate on the utility of computer science instruction in higher education, usability (nee HCI) has emerged to challenge software engineering as *the* relevant discipline in the design and development of information systems in an organisational context. This challenge has proceeded on two fronts. On one level a theoretical shift in computer science curricula has been advocated. For example, Kling [1] has argued that usability, 'one of the most dominant of the diverse trends in computing', requires human and social, rather than mathematical, theoretical foundations in computer science research and teaching (through what he calls the field of 'Organisational Informatics'). On another level, empirical models and cases have advanced the mechanics of usability instruction. The Association of Computing Machinery's Special Interest Group on Computer-Human Interaction has developed a HCI curriculum (ACM SIGCHI [2]), Perlman [3] has described the design and delivery of a User Interface Design course based on the SEI curriculum model and Nielsen and Molich [4] have shared their experience teaching User Interface Design to MSc Computer Science students in Denmark.

The case study presented here belongs firmly to the latter category. It follows closely the structure and presentation of the Nielsen and Molich case but differs in its use of Visual Basic as the prototype and build tool. Visual Basic allows programmers to create applications based on standard MS Windows features (such as dialogue boxes, command buttons, pull-down menus and

selection lists) in a fraction of the time taken by traditional programming methods. As an instructional tool in User Interface Design, Visual Basic compares favourably with Hypertext systems and easily out-scores static drawing packages. It is also intuitive, interactive and fun.

Background

The User Interface Design course described is one third of a non elective, twelve week, module in Information Technology taken by students studying for a Diploma/MSc in Information Management in the Division of Information Management, The Queen's University of Belfast. Prior to undertaking the course in UID, all students had been instructed - by other members of staff - in DBMS (5 weeks), Knowledge Based Systems (2 weeks) and introductory programming using Visual Basic (one full week of lectures and practical exercises). The UID course consisted of 32 hours of study, made up of four 2 hour lectures, four 2 hour practicals and an assessed project at 16 hours. This project, which accounted for 30% of the IT module mark, was an individual effort designing a user interface.

Although the Diploma/MSc is a conversion course aimed at non-computer science and non-information management graduates, the 27 students enrolled were reasonably computer literate. A third had a home computer, all but one declared knowledge of DOS and Windows and over half claimed a knowledge of Unix. More than half claimed knowledge of the following languages - Modula, Fortran, Cobol and Pascal - and 10 students had written a program that was used by others. Students spent an average of 10 hours per week using a computer during term and 8 hours per week out of term.

The course was designed around computer-based exercises in heuristic evaluation, interface design and the design of online help. Visual Basic 3.0 Standard Edition was used, running over Novell Netware 3.1. The core texts were Nielsen [5] Usability Engineering and Morris [6] Visual Basic.

Heuristic Evaluation

Getting students to critically evaluate a commercial application package is a valuable first exercise. It develops heuristic evaluation skills needed for later design exercises and, importantly, demonstrates that even professionally produced software has usability problems. The practical involved each student writing an evaluation report on the usability of Wordstar 2000 v3.5 and Wordperfect v 6.0 using the WP package being evaluated. Evaluators were asked to cross reference usability problems found to the usability principle(s)/guideline(s) violated. These guidelines had been discussed in-class and a paper exercise in heuristic evaluation completed. Also, each student had, as a reference, 26 heuristic evaluation guidelines derived from Nielsen's book.

Twenty one students took part in the practical, each producing two reports giving 42 reports in total. These reports were assessed on the basis of number of usability problems found and quality of report.

Most students were familiar with Wordstar (having used it for other course work) but were using Wordperfect for the first time. Unsurprisingly the Wordstar evaluation was completed first and to a higher standard. Clearly some students struggled with Wordperfect due to lack of knowledge or lack of time or both. A few even admitted to writing their Wordperfect evaluation report using Wordstar! In addition, a number of network implementation problems were encountered when using Wordperfect, including printing problems, file transfer problems and missing bits from the tutorial. Some of these problems led to incomplete or incomprehensible Wordperfect reports.

The majority of reports attempted to cross reference the usability problem found to the usability principle violated, although not always correctly and often in a very general manner. Many reported problems were vague, or had previously been reported in another guise or when checked by the instructor did not exist. One or two reports degenerated into product function descriptions. Finally, a number of students remarked that, either they found it difficult to evaluate Wordstar because they were familiar with it, or that they found it difficult to evaluate Wordperfect because they were unfamiliar with it. This highlighted the need to re-emphasise in class that, in the real world, heuristic evaluation and usability testing involve a mixed bag of evaluators/testers with varying levels of skills and knowledge (i.e. know the user).

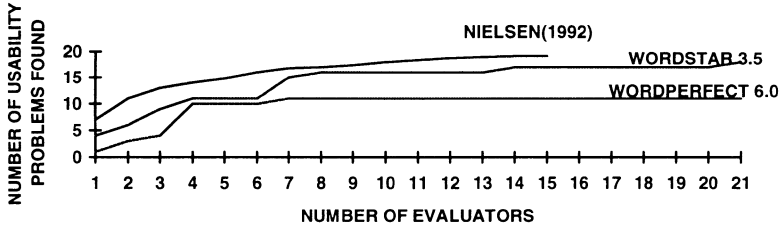


Figure 1: Usability problems found as a function of the number of evaluators

Figure 1 shows usability problems found by heuristic evaluation as a function of the number of evaluators. The top curve line shows the average results from six studies discussed by Nielsen [7] in which single evaluators found only 35% of the usability problems whereas 5 evaluators found 75% of the problems. He concluded the optimum number of evaluators in these projects was between 3 and 5 (the exact number of evaluators will depend on specific cost benefit analysis and the criticality of usability).

For this exercise, the evaluation reports were marked in random order, *new* usability problems recorded and the cumulative total plotted against each report/evaluator. The flattening of the Wordstar and Wordperfect curves after the seventh evaluator is noticeable, particularly with Wordperfect, reflecting the poorer quality of reports on this product overall. The steep rise in the Wordstar curve at the seventh evaluator and the Wordperfect curve at the fourth evaluator indicate a particularly good report (from an able student) recording new usability problems that were subsequently to be reported by other evaluators.

Interface Design

Students learn most designing and testing their own programs. This practical involved the design of a (hypothetical) multimedia application to be used by the Division as a promotion at school open days. As this was an exercise in interface design rather than programming, students were encouraged to concentrate on the main screens with some indication of the functionality lying behind two or three of the main menu options selected. In the event, this exercise was extended to cover two 2 hour lab sessions and formed the basis of many project submissions. In an effort to encourage creativity, students were not given a specific house style to follow but a set of guidelines derived from the course texts and covered in lectures.

Fundamental omissions were found in final designs such as no escape / exit, no keyboard shortcuts and unhelpful error messages. It is not certain that greater direction would have improved matters. Nielsen and Molich (1989) report similar oversights in design when students were asked to extend an existing system and given hypothetical in-house standards. A small number of students were able to edit standard icons or create their own using the ICONWRKS sample application provided with Visual Basic. These students experienced difficulty in locating this application and in finding out the range of icons supplied. To this end, two of the many forms and routines written by John Clark Craig [8] allow the user to easily browse through icon directories and display bitmap, metafile or icon image files from any directory or drive. Icons are worth spending some time on in the lab., not only because they add much to the interface but because their design and usage can be directly related to text book discussions and in-class paper prototype exercises.

Online Help

Help and documentation is an excellent finale to a course on User Interface Design. An interface within an interface and a microcosm of the entire course, help and documentation can be a sound basis for course assessment. The final practical and project were therefore intended not only to introduce students to the importance of good help and documentation but to consolidate the broader issues of iterative design, heuristic evaluation and user testing.

Students were asked to design and develop an online help facility for one of their own programs. Effectively this was a choice between the application completed for the programming course or the multimedia application and most students chose the latter. In an attempt to evaluate the process of design as well as the product, students were also asked to produce a written report of their experiences. This was to include a preliminary heuristic evaluation of the online help in a commercial application package.

Assessment of the project, which carried 30% of marks for the IT module, proved difficult. A simple check list of do's and don'ts seemed the most objective and a marking schedule containing 41 usability principles was drawn up. These were grouped under 5 major marking criteria and weighted - Message Boxes (20%), Menus (30%), Screen Design (10%), Written Report (30%) and Discretionary Mark (10%). (A Message Box is a single statement in Visual Basic which allows the display of a temporary dialogue box with any included line of text). In retrospect this method proved over simplistic and unfair. A scheme which awards marks for the inclusion or exclusion of certain items such as menus, check boxes and control buttons favours certain applications and interfaces more than others. In this case, the projects developed under the programming course were not as well suited to the addition of online help and attempts at improving the interface were less successful. This is reflected in the allocation of marks with students who developed the multimedia app. obtaining a higher average mark (52.5 compared with 41.5). Other deficiencies in the scheme became apparent when marking. Insufficient weight had been given to creativity in design (only a 10% discretionary mark) and some good innovations were not rewarded. Two criteria were found to be inapplicable to any of the projects being assessed, effectively reducing the maximum possible mark by 6%. Finally, it was clear that the criteria itself was not well enough explained to students before the project began.

Course Evaluation

A course evaluation questionnaire based on that used by Nielsen and Molich (1989) was given to students at the final class. Twenty questionnaires were returned, giving a 74% response rate. A summary of responses is shown in Table 1. As with the Nielsen and Molich survey, students liked the practical orientation of the course. 90% of respondents felt they learnt something from doing the practicals and only 5% felt that the course was too practical in nature. However, in answer to a further question, 50% felt that their knowledge of Visual Basic programming was too basic for the work required in User Interface Design (this probably refers to knowledge of Visual Basic rather than knowledge of programming, as the amount of coding required for the exercises in UID was minimal).

Table 1: Course Evaluation

	SD	D	N	A	SA
The course was too demanding and difficult	20	25	35	15	5
Too little was demanded of students	20	45	25	10	0
The course was too theoretical	5	60	20	15	0
I learnt something from doing the practicals	0	10	0	70	20
The course was too practical in nature	10	60	25	5	0
I would consider taking further courses in UID	20	20	25	25	5
I would like to take a full module in UID	15	20	20	25	20
% response; SD=Strongly Disagree, D=Disagree, N=Neutral, A=Agree SA=Strongly Agree					

Pedagogic Issues

Assessment

The use of a simple check list to evaluate interface design is attractive but needs great care. In this case, the method was bias towards a particular design and poorly rewarded creativity. Unease with the method used was compounded by final results which showed the average UID project mark to be much lower than other project marks for the IT module. Of course, other factors must be taken into account, (new course, new subject, new software and end of semester timing) but assessment method was, and is, important. Inflexible and unimaginative design is a heavy price to pay for pseudo objectivity achieved through set marking criteria. One solution may be self and peer assessment which nicely reinforce instruction in heuristic evaluation and user testing and facilitate the collection of subjective judgements on good design.

Theory versus Practice

A set of lectures based on Nielsen's Usability Engineering and hands-on practical sessions using Visual Basic is an excellent introduction to the practice of usability, but not enough. Students need some background in the theoretical aspects. Classroom discussions here would have benefited from shared understanding of basic concepts such as interface, dialogue, user models and metaphors (formal methods such as GOMS and TAG need only be discussed as a reference for the more practical methods covered). Although covered briefly in lectures and through reference to Paul Booth's [9] excellent introductory text on HCI, student work revealed inadequate understanding of these fundamentals. This said, Nielsen's assertion that all usability courses should be firmly based in the lab is worth repeating. Some practical exercises can be performed in class, with small teams producing paper prototypes for example, but it is the authors experience that both instructor and the student recognise this as being second

Transactions on Information and Communications Technologies vol 7, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

best. The combined effort required to give credibility to these exercises (and avoid, what one colleague has called, 'the Playschool effect') is difficult to justify when better, richer, development environments are available in the lab. Classroom sessions are best used to provide the framework and to discuss the increasing amount of good case material coming out in this area (see for example, Wiklund, 1994 [10]).

Freedom v Control

Ideally a student should be free to develop an application of their choice, within broadly defined criteria. In practice this creates difficulty for the instructor. Students who used applications written for the programming element of this module as the basis for user interface design, suffered due to the nature of the problem tackled. Many students presented a project based on a questionnaire survey of consumer behaviour which was itself poorly thought out. This contributed to an inflexible and unexciting user interface. Moreover, the absence of specific design criteria (in the form of an existing design base or house style) maximised freedom but resultant variability complicated assessment.

Scheduling

The best time to introduce programmers to usability is at the beginning of the module. This year, we were guilty of a common approach in computer science - we stuck it on at the end. Starting with usability is not only pedagogically sound, it maximises the strengths of a highly interactive tool such as Visual Basic with which students can develop quite sophisticated interfaces with little or no coding and then add the functionality. Next year our module will begin with an introduction to Visual Basic and usability, followed by programming instruction and finish with more advanced usability issues. This approach should provide a gentler introduction to programming (with coding following interface building) and hopefully avoid student concerns about lack of programming knowledge (expressed in the course evaluation).

Constraints

36 hours of study, only 8 of which were in the form of scheduled lab practicals, is a pint pot in which to squeeze the quart of usability. The most serious omission, despite best intentions, was that of user testing. While heuristic evaluation does offer the insights of self examination, nothing beats someone else commenting on your own design. An exercise in thinking aloud testing [11] - a simple and inexpensive method of user testing - had to be left out due to time constraints. Inclusion of some form user testing is a must, the student body is an excellent source of test users, although testing involving actual users can be more problematical. Given sufficient time, it is desirable to include discussion of advanced issues such as next generation user interfaces and computer supported cooperative work. The latter is particularly relevant and can map directly to a number of practical exercises, most obviously some form of thinking aloud

testing in groups. Group testing and group design will become a feature of future courses; this will bring new challenges in assessment.

Conclusion

While student feedback, measured as quantitative and qualitative responses in the course evaluation, was overwhelmingly positive, it has proved difficult to measure the impact of the course on *programming performance* - due to the variation in assessment methods used. However, a number of observations can be made. There was little evidence of an across the board improvement. Average marks for the programming and the UID project were the same (48) and, of the 22 students who submitted both projects, 9 improved, 1 stayed the same and 12 fell behind. Of those who improved, two students moved from the lower quartile of programming marks to the upper quartile of UID marks, one of whom obtained the highest UID score in the process! Disappointingly, three students in the lower quartile of UID scores had obtained very much better programming marks, perhaps reflecting end of semester overload. Overall, individual programmers appear to have benefited from usability instruction but these results do not support a correlation between user interface design instruction and programming performance.

References.

1. Kling, R. Organisational Analysis in Computer Science, The Information Society, 1993, 9, 2, 71-87.
2. ACM SIGCHI, Curricula for Human-Computer Interaction, Association for Computing Machinery, ACM Order N0. 608920, New York, NJ. 1992.
3. Perlman, G. Teaching user-interface development, IEEE Software, 1990, 7, 6, 85-86.
4. Nielsen, J. & Molich, R. Teaching user interface design based on usability engineering. ACM SIGCHI Bulletin, 1989, 21, 1, 45-48.
5. Morris, S. Visual Basic Step by Step, Newtech, Oxford, 1992.
6. Nielsen, J. Usability Engineering, Academic Press, London, 1993.
7. Nielsen, J. Finding usability problems through heuristic evaluation, in Proceedings of ACM CHI'92 Conf, pp 373-380, Monterey, CA, 3-7 May, 1992.
8. Clark-Craig, J. Microsoft Visual Basic workshop, windows edition, Microsoft Press, Redmond, WA, 1993.
9. Booth, P. An Introduction to Human-Computer Interaction, Lawrence Erlbaum Associates, Hove, East Sussex, 1989.
10. Wiklund, M. (ed). Usability in Practice, How Companies Develop User-Friendly Products, Academic Press, London, 1994.
11. Nielsen, J. Evaluating the thinking aloud technique for use by computer scientists, in Advances in Human-Computer Interaction, ed. Harston, H.R., and Hix, D., Vol. 3, pp 69-82, Ablex, Norwood, NJ., 1992.