Usability Measurements in an Undergraduate Programming Course

William J. Adams
Dept. Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996
(914) 938-5559
adams@eecs1.eecs.usma.edu

Bernard J. Jansen
Dept. Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996
(914) 938-3233
jjansen@acm.org

Richard Zoller
Dept. Electrical Engineering and Computer Science,
United States Military Academy
West Point, NY 10996
(914) 938-5995
zoller@exmail.usma.edu

Abstract

Usability is an often neglected component of software design. Usability principles are usually relegated to advanced courses in Human Factors or User Interface design, without consideration for the impact they have on program functionality. Using an analysis methodology, usability principles are identified for integration into an existing computer science course. The key feature of this integration is that it must take place without displacing any material currently taught in the course. In other words, students are being implicitly taught usability through their normal instruction in programming functionality. The experiment performed on an introductory level computer science course showed a significant improvement in usability in student projects without impacting functionality. This paper describes the analysis and methodology, the usability principle selected and the impact they had on the students' projects. The results of this study indicate that usability instruction can be introduced into an existing computer science course, usability of student programs can be improved, and no existing material need be displaced.

Keywords: Empirical studies, HCI education

Please Cite: Adams, W. J., Jansen, B.J., and Zoller, R. 1998. Usability measurements in an undergraduate programming course. Software Engineering Conference. Las Vegas, Nevada.

Introduction

Usability and functionality are integral design components of any successful project, and it is unrealistic to treat them separately. Many undergraduate computer science courses focus primarily on the functionality of programming projects. Usability principles are usually introduced in advanced classes late in the curriculum, if at all. Unfortunately, this approach teaches students that usability is separate from functionality. With the emergence of compilers that allow easy access to graphic user interface (GUI) libraries, it is now possible to incorporate usability principles into the curricula without displacing currently taught concepts. The goal, therefore, is to integrate usability design principles while neither replacing courses, deleting material, nor adding new courses. The motivation for doing this is three-fold. First, there is rarely room available in the students' schedule for an additional engineering course. Second, course directors are already forced to make hard decisions on what material they can teach and what has to wait for another course. There is simply not enough time to teach other. Third, a separate usability class does not reinforce usability as an integral part of the design process.

The first section of this paper will review usability instruction discussed in current computer science literature and follow with a discussion of how to seamlessly integrate usability into an existing course. The second section will discuss an analysis methodology to chose which usability principle to integrate. In the third section, the paper shows the results of an empirical study performed to display the results of usability integration. Finally, the paper discusses the implications of using the analysis methodology in other classes to continue usability integration.

Related Work

The manner of communication between human and computers continues to evolve. Because usability in the computer science area studies humans and computers in communication, it draws from supporting knowledge on both the computer and the human side. It is important to organize the curriculum in a changing field with an understanding of the forces shaping that field. This is done in an attempt to ensure that its concepts are not out of date. In other words, the curriculum must build the foundations for its graduates. From this foundation, its graduates can develop their own understanding of the future.

The Problem

Due to the turbulent nature of the usability field, there is little agreement on how to incorporate usability into an existing computer science curriculum. A lesson learned by many engineering disciplines is that design problems have a context that can invalidate any narrow optimization of one specific perspective (ACM, 1997.) The proper study of usability exists within the larger paradigm of an engineering design methodology.

There are several approaches that attempt to address this aspect of usability instruction and curriculum development. Studies show that usability is a critical ingredient in project development and should be incorporated into the design process (Lundel and Notess, 1991.) Various academic institutions have attempted different approaches in introducing usability into the course work. Most schools have simply added one or two usability courses to the curriculum offered within an engineering department. Other universities have taken a multi-disciplinary approach linking courses within different departments (Foley, 1991.) Finally, another approach is to introduce several usability courses and develop a separate field of study focusing on usability within a particular engineering department (John and Morris, 1993.)

All three approaches have certain economic and political costs that many times make them unworkable. For example, if a curriculum introduces a mandatory usability course, which course currently in the curriculum is dropped? If no courses are dropped, where does the financial support come from to add the usability courses to the course load? What free elective is dropped? Can other courses survive with the decreased enrollment? More importantly, all three approaches present the message that usability is somehow separate from the other engineering areas or at best is an add-on to be considered once the "real" work is done.

This situation can be avoided since usability has an inherent relationship to almost all other engineering areas. Since usability, in computer science, focuses on the computer system, it includes both the computer and the human. Due to this fact, it has the advantage of being easily incorporated into existing computer science courses. Additionally, usability has many different levels of complexity. One can tailor the introduction of principles to the sophistication level of the students (i.e., simpler principles at the freshman level and complex or subtler principles at the graduate level). This introduction is directly related to the interlocking relationship of usability and project design.

Our proposal overcomes many of the economical and political disadvantages of the other methods. Of course, one must convince the individual instructors that the introduction of usability principles will benefit their courses. We have attempted to ensure that the introduction of usability principles is beneficial to the course by building on the concepts already existing in the course. We have chosen to test our proposal with actual introductions of usability principles into existing computer science courses. From these and other tests, we hope to build a theory of usability integration. This method of deriving usability principles from actual concepts, products, or prototypes has historical roots in the engineering community. These include the conceptual Memorex (Bush 1945), the Sketched (Shneiderman, 1998) and the work of Engelbart (Engelbart, 1988.)

Proposed Solution

The solution is not to teach usability but to teach user-centered design. Rather than teach a separate block of usability instruction, integrate usability and functionality seamlessly into the course’s topics. The particular user-center design methodology will vary course to course, incorporating only those usability design principles that bring value-added to course. The usability principles introduced are those that reinforce the concepts and techniques that the course is currently teaching. For example, in teaching an introductory programming course, the user-centered design would focus on the use of selection structures as a means of error checking for the user. Repetition structure could be introduced as a means of developing a menu or user interface.

Continuing, a major objective of most introductory computer science course is to teach control structures. Control structures are perfect for error checking and human engineering of output statements. In user-centered design, one can teach control structures with usability as a reason for implementing them. The advantage is that the student immediately sees the benefit and the product is better.

No new concepts are introduced, but rather the primary functional goals of the course are focused on the user. The particular usability principles behind the design need never be introduced.

The advantage of this user-centered design approach is that usability is part of the design process at the earliest stage of a student's education, resulting in computer scientists that can design programs that people can use without treating usability as an issue separate from functionality. It also overcomes the complexity issue by "black boxing" the usability principles within the course's current framework. Finally, it brings added value to the course's content by only introducing those usability principles that enhance the core content of the course.

The perfect solution to this dilemma would be to analyze and alter the entire curriculum to integrate usability. This way, the usability and user-centered design aspects would be consistent and would leverage the students' increasing experience base to enhance their understanding of Human-Computer Interface issues. This is not, however, feasible without some proof that there can be seamless union of usability and functionality in a course without either adding more time or taking away any material.

In light of this fact, this project intends to test this theory that usability can be taught in an introductory computer science course without sacrificing any of the instruction in functionality. The course currently has two main goals. The first goal is to familiarize students with the fundamental concepts of computer science, while the second goal is to develop students' proficiency in an engineering problem solving and design methodology. This methodology currently teaches a software development process and stepwise refinement to improve the functionality of programs. A user-centered design approach may reinforce both the software development process and the step-wise refinement process in the use of input and output statements.

The particular methodology is called a **SPOT** analysis. This analysis focuses on a course's **S**chedule, student **P**opulation, **O**bjectives, and **T**ools. Schedule is the amount of time available in a course, measured in hours or lessons. Population is the are the characteristics that define the majority of a course's students. Objectives are the purposes or goals of the course. These are usually outlined in the course objectives and are supported by one or more lesson objectives. Tools are what the students use to implement the course or lesson objectives and can be a programming language, hardware, etc. The SPOT methodology focuses the usability instruction to a user-centered design process that is seamlessly integrated with the functionality of a course.

The goal of the SPOT analysis is to capture the critical aspects of a course in order to identify which usability principles or techniques are applicable to that course. One of the major obstacles preventing usability instruction in computer science courses is the resistance to introducing new and seemingly unrelated material. The SPOT methodology accepts this assertion and identifies usability aspects that directly related to a particular course. Using the SPOT methodology also addresses the other issues of usability principles being too difficult to introduce and courses having too much material already. The outcome of the SPOT analysis is a usability technique or principle that directly relates to a course and adds no or little new material. Additionally, the usability technique is understandable to a course's students because it is already within the domain of the course material. The goal of a SPOT analysis is to get usability improvement beyond the aspect of whatever technique was introduced. Note that the SPOT analysis does not imply a change in grading policy, schedule, course objectives, student perquisites, or course material. Nor does SPOT separate usability from functionality.

The Experiment

This experiment focused on the issue of improving the usability of computer science student programs without a significant change in course material and no change in teaching method. The experiment introduces usability instruction into an existing computer science course based on the SPOT methodology. The experiment then measures the effect of the instruction on the usability of student programs.

Hypothesis:

Students will design and implement programs that exhibit better usability if a course implements results from a SPOT analysis than if the course does not.

Method

Subjects

There were 252 student programs evaluated in the experiment. There were 114 student programs from sections without the SPOT implementation. There were 138 student programs from sections with the SPOT implementation. The programs were the final design project from the programming portion of the course. They represented both a breadth and depth of the students' knowledge about the course subject matter.

Course

For the course, the experiment utilized CS105, *Introduction to Computing*. This course is mandatory for the freshman class at the United States Military Academy. The average freshman class is approximately 1100 students. About 50% of the freshmen population take the course in a given semester. Beginning in their junior year, all students must take a 5 course engineering sequence. Therefore, CS105 serves as the initial technology and engineering design course for all students. Since all math, science, and engineering departments depend on CS105 for introductory design and programming material, the course is extremely scrutinized at the Academy-level for course content. This makes CS105 a perfect choice for this experiment because this is exactly the situation that many computer science instructors cite for not integrating usability instruction, i.e., they cannot add new material. The course is divided into three phases, a computing concepts phase, an engineering design phase that focuses on developing software programs, and a problem solving phase using applications.

SPOT Analysis

This experiment focused on the engineering design phase of CS105. This phase is 20 lessons of the 40-lesson course. So, the Schedule aspect of the SPOT analysis was about one-half of the course. The Population of the course is freshmen students who are randomly assigned to sections in the course. Computing and technology experience varies widely among the course's student population. The Objectives during this phase of the course are the design principles of selection, iteration, and modularity. The Tool utilized during this phase of the course is the Ada 95 programming language, a general purpose programming language. Table 1 summarizes the results of the SPOT analysis for CS105.

SPOT Analysis	Results
Course Schedule	20 lessons
Course Population	First Year Students, Varied Computer Expertise
Course Objectives	Selection, Iteration, and Modularity
Course Tools	Ada 95

Table 1. SPOT Analysis of CS105 With Results.

Using the results of the SPOT analysis, any technique that addressed both functionality and usability had to be simple to incorporate in 20 lessons to novice students, had to directly relate to selection, iteration, and modularity, and had to be programmable in Ada 95. These results led to the adoption of exception handling as technique to introduce usability and reinforce functionality.

Students are already taught to use Ada 95 subtypes in their programs to constrain user inputs to acceptable values. Unfortunately, when a value is out of range or badly formed, the student's program "crashes," leaving students wondering why they would want to design a program that fails when the user inputs an unacceptable value. To prevent program "crashes," Ada 95 has built-in error checking mechanisms, called exception handlers. Without invoking error handling subprograms by explicit calls, exception handlers allow a program to continue normal execution when a condition or event occurs that might be considered exceptional. For example, an exception handler could be used to prevent a student's program from "crashing" when a user inputs a value that is out of range for the declared subtype. Exception handling may be a valuable tool to introduce usability principles in a manner that reinforces the concepts of the course. Exception handlers are not currently used in the course due to their complex nature.

Although exception handling can be confusing, it can be "black boxed" for early lessons. As the course progresses, the exception handlers can be "opened up" to teach important programming functionality concepts, such as, scope, visibility, parameter passing and control structures. The introducing of exception handling enhances usability and provides a mechanism to introduce repetition and selection statements, procedures, scope, visibility and file input and output. As such, it is a natural addition to the course. It adds little to no burden for the instructor since exception handling provides a built-in example for many functionality aspects of CS105 that the instructor must introduce already.

Design and Procedure

All 252 computer programs were evaluated on usability aspects by the authors in a blind evaluation. Blind in that the evaluators did not know the students' name, grade or whether the student was in a section that received the exception handling instruction. There were 118 programs from sections that were not introduced to exception handling and 134 programs from sections that received the exception handling instruction. It is reasonable to assume that the sample size of 252 satisfactorily represents the student population of 1100. The sample size is approximately 23% of the student population. However, a voluntary re-sectioning occurs during phase one CS105, and this re-section could have an impact on the representatives of the sample population.

The evaluation utilized an instrument based on usability design principles of Norman (Norman 1988) and Nielsen (Nielsen 1993). The instrument was a 10 item questionnaire. Each item focused on a basic usability aspect and was evaluated on a 10-point Likert scale. For example…..

Results

An Analysis of Variance (ANOVA) was conducted on the usability score for each program. An ANOVA is concerned with the statistical relationship between one or more predictor variables and a response variable. This experiment analyzed the response in usability score against whether exception handling was introduced or not. The hypothesis was supported. There was a marked difference in usability scores based on instruction, as

Figure 1 illustrates. The group of students that received the exception handling instruction scored much higher than those students that did not receive the exception handling instruction. In addition to instruction, the instructor and cognitive ability of the individual student could have impacted the response variable. Therefore, instructor and student's overall grades were also included in the ANOVA model. Grades were used as a measure of a student's cognitive ability. There was no correlation between usability scores and a particular instructor, but there was a correlation between usability scores and the overall course grade. Table 2 presents some of the relevant results from the ANOVA. Tables 3 and 4 present further descriptive statistics from this experiment. Table 3 presents the statistics of usability score and grades relative to instruction. Table 4 presents statistics of usability score and grades relative to the instructor. As shown, there was also no effect on either usability scores or grade relative to instructor.

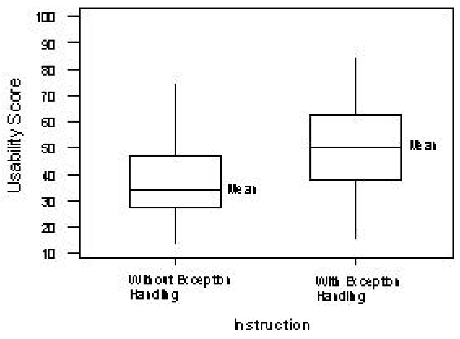

Figure 1. Mean Usability Score as a Function of Instruction.

Source	DF	F	P
Instruction	1	52.77	0.000
Instructor	1	0.04	0.849
Grades	1	13.02	0.000
Error	248		
Total	251		

Table 2. Analysis of Variance for Usability Score

Variable	Instruction	Mean	St Dev

Usability Scores	Both	44.65	17.01
Usability Scores	1	37.14	13.70
Usability Scores	2	51.26	16.94
Grades	Both	85.82	5.47
Grades	1	85.67	6.00
Grades	2	85.94	4.98

Table 3. Descriptive Statistics Relative To Instruction

Variable	Instructor	Mean	St Dev
Usability Scores	1	44.30	17.74
Usability Scores	2	44.94	16.45
Grades	1	84.51	5.568
Grades	2	86.91	5.173

Table 4. Descriptive Statistics Relative To Instructor.

Discussion

The evaluation of the labs and the ANOVA yielded some interesting and significant results. There was a significant relationship between instruction and the degree of usability that students exhibited in their programs. The introduction of exception handling increased the mean usability score by over 14%. Considering that exception handling introduced no new functionality concepts and was never a determinate on grades, this is a substantial improvement in the design process.

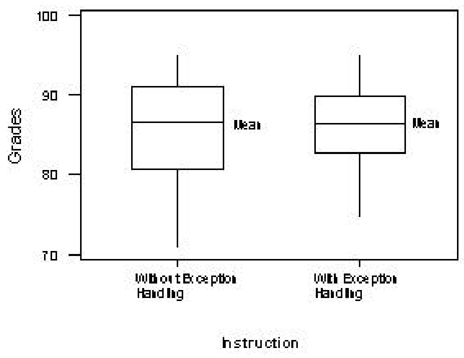

Figure 2. Mean Grades as a Function of Instruction.

From the analysis, grades were also a contributing factor to the usability scores. If grades are a reflection of cognitive ability, as one would hope in the teaching profession, this appears reasonable and expected. However, from a statistical standpoint it raises concerns. For example, the group undergoing the new instruction could just be a smarter group of students. If true, this higher cognitive ability rather than teaching method could be the significant factor causing the increased usability scores. However, it appears that this is not the case. The two groups are of almost identical cognitive ability based on grades. Figure 2 illustrates that the two groups of students received reasonably the same grades, with the means between the two groups being almost identical. Again, Table 3 presents the complete statistics for grades by term and Table 4 presents the complete statistics for grades by instructor.

Conclusion

In conclusion, this experiment has demonstrated that SPOT analysis allows usability to be seamlessly integrated with functionality in an undergraduate computer science course.

Apparently the SPOT analysis can lead to significant usability design increases within an existing computer science course. More importantly, this increase comes with no changes in teaching method or course content or goals. Instead, it builds on the existing material within a course to achieve the usability increases with a user-centered design framework.

Although, the SPOT analysis was only applied in one class, it is general enough to be applicable to many other computer science courses. Since the focus of SPOT is on the design process, it should be transferable to other engineering courses outside the computer science discipline and in graduate courses. Given the proven efficiency of the analysis and the benefits of adding usability, the important next step is to consider where usability principles best fit into the curriculum as a whole. These are areas of future research.

References

ACM Special Interest Group on CHI Curriculum Development. ACM SIGCHI Curricula for Human-Computer Interaction. Referenced at http://www.acm.org/sigchi/cdj. August 1996.

Adams, William J. and Jim Jansen. *Integrating Usability Design Principles into an Existing Engineering Curriculum*. Proceedings of the American Society of Engineering Educators 1997 Conference. Milwaukee, WI, June 1997.

Bush, V. *As We May Think.* The Atlantic Monthly, Vol. 176. no. 1, July 1945, 101-108.

Center for Information Systems Agency. Human-Computer Interface Style Guide. Version 1.0 12 Feb. 92. Cameron Station, Alexander, Virginia.

Engelbart, D.C. *A Research Center for the Augmentation of Human Intellect*. In Grief, I. (Editor), Computer-Supported Cooperative Work, Morgan Kaufman, Palo Alto, CA (1988), 81-105.

Foley, J., Mitchell, C., Waler, N. *Human-Computer Interaction Research at Georgia Institute of Technology*. Proceedings CHI'91, 61-62.

John, B. and Morris, J. *HCI in the School of Computer Science at Carnegie Mellon University*. Proceedings INTERCHI'93, 49-50.

Lundel, J. and Notess, M. *Human Factors in Software Development: Models, Techniques, and Outcomes*. Proceedings CHI'91, 145-150.

Marcus, Aaron. *The Future of Advanced User Interfaces in Product Design*. TRON Project, 1992 Symposium: 14-20.

Nielsen, Jacob. Usability Engineering. AP Professional: 1993.

Norman, Donald. The Psychology of Everyday Things. New York: Basic Books:1988.

Shneiderman, B. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Reading, MA: Addison-Wesley. 1998.