

# Where SE and HCI Meet: A Position Paper

Mary Jane Willshire, Ph.D.  
University of Portland  
willshir@up.edu

## Abstract

*One way to remedy the gap that currently exists between software engineering and human computer interaction is to expose undergraduate students to the ideas, concepts, processes, methodologies and jargon of each field of endeavor. In teaching both software engineering and human computer interaction courses to undergraduates, I have found places where topics from one discipline can be inserted into the course material for the other to emphasize the connections that exist between the two. My hope is that by showing these bridges to students, they will enter the working world or graduate studies with an increased sensitivity to the issues. Perhaps this sensitivity will manifest itself into changed practice. This position paper discusses the approaches that I use to teach students.*

## 1. Introduction

There are many challenges in teaching students with little or no industry experience either software engineering or human computer interaction. However, I feel that it is vitally important to introduce undergraduate students to the worlds of software engineering, human computer interaction, process, documentation and their interdependences. Cheston and Tremblay who point out that software engineering concepts can become part of introductory computing courses [2] also discuss the difficulties faced by those of us who attempt to do so. Typical undergraduate students have no concept of the scope of work or amount of effort involved in building a large, complex system. The ideas of systematic processes and documentation consisting of anything other than comments in their code are totally alien to them. In their minds, if they write a program and it compiles and runs, then they have built a “perfect” software system. Their idea of an interface is at best a menu driven, text based interface or maybe some

Java graphics with simple buttons and sliders. Their programs are mostly written by one person, and tested by that same one person (or maybe their instructor or grader). Given these challenges, I have developed some techniques for teaching traditional undergraduate students these subject areas and especially some bridging points between software engineering and human computer interaction. This paper is an overview of those techniques—approaches that have worked and approaches that need more development.

## 2. Commonalities

What are the commonalities between the standard way that one teaches software engineering and the way that one teaches human computer interaction? For many, no commonality exists. However, if one looks, there are indeed places where in each course one can cross-reference to the other. Table 1 summarizes some of these commonalities and how I have introduced them in each subject area. For instance, every software project should have a detailed requirements document. How is this any different from the requirements that an HCI person would need? Or what should be added to the requirements document to reflect the users needs?

**Table 1. Commonalities**

SE in HCI	HCI in SE
Requirements for usability	Usability included in requirements
Use cases—task analysis	Use cases from the users
Detailed scenarios for system tasks	User-focused detailed scenarios
Role of prototypes in system development	Prototypes of user interface
Usability testing	Usability testing
Software process models and work products	User-centered development

Another commonality is the role of use cases and scenarios. What are these but descriptions of how a system will be used by the users—hence, these can form the basis for usability studies and the setting of usability goals. A third, though somewhat weaker commonality, is prototyping the system. A simple paper (or low fidelity) prototype is sufficient for early phases of usability testing. It can then be used by the developers in building the screens for their functional prototypes. The rest of this paper discusses in detail how these commonalities are brought forward in teaching software engineering and human computer interaction courses.

### 3. Teaching SE in an HCI course

When one is teaching a course with HCI as its major topic, one can use either a psychology-oriented approach or a software development approach. If one chooses a psychology-oriented approach, there are large selections of excellent texts to use for the course. Many of these texts are targeted to the graduate level, but can be used by an undergraduate course by selecting some chapters and omitting others. This approach builds on the human factors that have been studied for years by specialists in HCI. While we have much to learn from their efforts, this approach tends to ignore the software engineering issues entirely. For traditional undergraduate students with the mistaken idea that all of computer science is writing small programs for their own use, there is little understanding of or appreciation for this body of knowledge. I have tried this approach with undergraduates and felt that the course was a dismal failure.

If, instead, one decides to try a software development approach then one immediately encounters particular challenges. The first such challenge is that there is a severe lack of texts written that target an undergraduate audience in HCI with a software focus. One such text is Hix and Hartson [3] but it assumes a software background that many undergraduates do not have. A newer text is that by McCracken [4]. The McCracken text uses web page design as its theme. This topic is understandable to undergraduate students and has great appeal. I used this text (in manuscript form) this past academic year and my students really liked it. The one downside is that the text does not in and of itself, present actual software engineering. However, it does emphasize a process-oriented approach to interface design. This provides a way for the teacher to bridge to the software development processes.

What I have done that works well with undergraduates is to select a text that is not too heavy on the psychology, supplement it with lectures and assignments on software process, software lifecycle issues and use software work products such as requirements documents, use cases, scenarios and test plans as parts of the course project. For example, students are told that they have to do a project to design an interface and that there are several deliverables for this project. The first stage in the project is to select the “product” for which they will develop an interface. They must write up a description of this product, that is, provide a requirements document for their product. Initially, I don’t call this assignment a “requirements document” but I tell them that they have to turn in a written description of the product, what it is, what it will do and so on. I gather these, read them and then start asking the students questions about their products. Based on these questions, they revise their descriptions. At this point, I point out that what we are doing is really eliciting the requirements for the project and writing a requirements document. That then leads into a discussion of what are requirements, how are they elicited, how are they used and where this fits into a development project.

After students begin to understand requirements, I then introduce the idea of use cases. Many of my students have never to this point in their studies considered the idea that some things are part of the system and some things are outside of the system. They easily understand that human users are not part of the system but provide inputs to the system; however, they do struggle with the concept of other external actors such as the network or a database. Unfortunately, an HCI class only has time to go so far down this path and for most undergraduates that means just working with humans as the only external actors in a use case diagram. Even so, this does make the students start to think about the interaction between the system and the users and makes them document how a user might want to interact with the system. Aha, now they are beginning to understand the context of a system and the use of a standard software modeling technique. Then I ask them what sorts of wrong things a user might do and have them do one or two of those as use cases. Recently, an article by Alexander [1] discussed this approach when taken to its extreme of hostile actions as “misuse cases”.

Frequently, students realize that their requirements are incomplete as they begin to identify their use cases by doing a task analysis. They may be surprised by this, which then leads to class discussion

about problems with incomplete or incorrect requirements. When the students have a good set of use case names in their use case diagrams, I then point out that these really don't have much detail and aren't all that helpful in designing the interface. So, from the use case diagrams the students move on to developing the scenarios for their use cases. For scenarios, the students write detailed interaction scripts for both correct and incorrect user actions (primary and secondary scenarios). They only have time to do this for one or two use cases. We discuss the value of doing a more complete set of scenarios, but typically, there isn't time in this setting to really expect the students to do so. Again, this introduces students to how one develops scenarios for the use cases, how one models user-machine interactions through a standard software modeling technique, and how this often causes the revisiting and revision of the requirements documents. The idea of revising something that they have already "finished" is another strange concept for many students. "You've already graded it, why should I redo it?" is a typical question. Well, my dear students, welcome to the world of iterative development.

By the time students have written scenarios for their interfaces, they think they are ready to start doing the design and truthfully, many have already done so in their heads if not on paper. However, I make them take one more step before going on to design. I have the students set up usability goals for their interface. I make them decide how good is good enough for their design. This is quite a shock to undergraduates. What do you mean, the users will make errors? No they won't, my interface will be so easy to understand that no one will make mistakes. I tell them that any software has to have acceptance criteria and they have to set the criteria for this product—how fast can a user be expected to perform some task? How many errors might a user make but still have acceptable completion of the task? And so on.

Setting usability criteria goes along with writing test scripts for later usability trials. Some students have some sense of testing software, but many feel that compilation alone is sufficient testing! They reluctantly set the criteria and start to think about testing. At this point, I usually relent and let them do their design and build a paper prototype for their interface. We discuss prototyping, its purpose and place in the development lifecycle and how it can be misused. We talk about different sorts of prototyping and why a paper prototype is often the best choice for interface design. We also discuss the

disadvantages to low fidelity prototypes versus higher fidelity prototypes.

At last, the big day arrives when the students have written a usability test script and have a paper, or low fidelity, prototype of their design ready for testing. With great confidence they launch into their first experience at having a user who is not part of the team (but is a class member) try to use their design. They are very surprised when their classmates make errors in executing the tasks or don't understand how to begin a task, much less complete it. Back to the drawing board—iterative development and an easy to change paper prototype make more sense now. A second round of tests is scheduled and the project concludes with a set of revised documents and a lessons learned paper submitted for grading.

To drive home the points on software process, I like to have the students do a second, more extensive project. The second time around, they really understand that each work product will feed into a next phase of development and that discoveries in later phases mean the revision of earlier documents. They also have a greater appreciation for the level of detail that is required in the earlier phases in order to have a successful design at the end.

For the second project, I usually ask the students to find usability trial participants who are not members of the class. Here they realize that outside users are not the same as "inside" users and that significant effort is involved in finding participants. This then opens up a discussion of why organizations often do not use HCI best practices in their software development efforts. Students report learning more from this second attempt than they did from the first project.

#### **4. Teaching HCI in SE class**

Teaching software engineering to undergraduates in some ways has fewer hurdles and in other ways has higher ones. Good textbooks exist [5] [6], students have written programs and have used various software systems. This means that the terminology is more familiar or at least often connects with their experience in some way. However, most undergraduate students have never thought of a process for development; they have never had to consider what goes into a requirements document; testing means compile the program and run it once; and documentation means writing a few comments in the code as an afterthought. This means they come into a software engineering course thinking that they already know all there is to know about software development.

The first hurdle is to make students aware of software process and process models. I include a user-centered development approach as part of this presentation. As I discuss the phases of a project, I ask the students to identify stakeholders for each phase. Since users of the final product are indeed important stakeholders, it is not difficult to repeatedly bring the user's perspective into the discussion. However, it is usually the instructor, not the students, who must identify this perspective. Students are not attuned to the user's needs. They simply want to get to the coding part of the process.

Throughout the software engineering course, I inject usability terminology into the software jargon as often as possible. When we discuss requirements, I ask the students to consider what should be in the requirements document that would reflect the usability levels for the product. I ask them how they would elicit this information, how would they document it and so on. When we discuss what is meant by software quality, I make sure that usability is on the list. Most software engineering texts (suitable for undergraduate courses) do not address these issues. Likewise, when we do use cases and their scenarios, I require the students to consider the users of the system. The next time I teach this course (Spring 2003) I will have students add diagrams (a low fidelity paper sketch) of interface components to the documentation of use cases for the primary and secondary scenarios as is done in the book by Schneider and Winters [7].

One thing that I have not tried is to ask software engineering students to design a usability trial. If I add this to the course, I will have to drop or reduce coverage of other topics. Currently, I talk briefly about usability testing and encourage the students to take the HCI course to learn more. Ideally, students would take both courses. Until recently, each course was considered a technical elective and rarely did students take both. Our new students are required to take the software engineering course and HCI is an elective. This gives me at least one chance to teach our undergraduates that users are important and must be considered throughout the software development process. Of the two courses, HCI and software engineering, I feel that the software

engineering course could use the most improvement in bridging the issues.

## 5. Summary

Teachers of undergraduate computer science students have the opportunity to introduce new ideas and concepts to open minds. Thus we can teach software engineering students the fundamentals of usability and teach HCI students the fundamentals of software processes. To be successful, the commonalities of each discipline must be identified and then those commonalities given a prominent place in the course structure. In this position paper, I discussed some of these commonalities and how I incorporate them as I teach the two courses.

## 6. References

- [1] Ian Alexander. "Misuse Cases: Use Cases with Hostile Intent", *IEEE Software*, IEEE Computer Society, v20, n1, January/February 2003, pp 58—66.
- [2] Grant A Cheston and Jean-Paul Tremblay. "Integrating Software Engineering in Introductory Computing Courses", *IEEE Software*, IEEE Computer Society, v19, n 5, September/October 2002, pp 64—71.
- [3] Deborah Hix, and H. Rex Hartson. *Developing User Interfaces Ensuring Usability Through Product and Process*, Wiley Professional Computing, New York, 1993.
- [4] Daniel D. McCracken and Rosalle J. Wolfe. *User Centered Web Site Design A Human Computer Interaction Approach*, to be published, Prentice Hall, 2003.
- [5] Roger S. Pressman. *Software Engineering A Practitioner's Approach, Fifth Edition*, McGraw Hill, 2001.
- [6] Stephen R. Schach. *Object-Oriented and Classical Software Engineering, Fifth Edition*, McGraw Hill, 2002.
- [7] Geri Schneider and Jason P. Winters. *Applying Use Cases, Second Edition*, Addison-Wesley, 2001.