COMPARATIVE STUDY OF HUMAN COMPUTER INTERACTION DESIGN AND SOFTWARE ENGINEERING

John D. Fernandez & Larry Young
Texas A&M University – Corpus Christi

ABSTRACT

Many computer science programs require students to complete software engineering and human computer interaction (HCI) courses. Upon graduation, these students join other software professionals in the field to contribute to the development community. However, the differences in the two approaches to developing interactive software are not addressed so that students leave the institution without an integrated view of the two methodologies. Professors at Texas A&M University – Corpus Christi teach courses in software engineering and HCI and assign students to community projects where students practice the principles they are learning and complete worthwhile products for real-world clients. This paper presents some of these experiences and compares the interaction design and software engineering methodologies. The conclusions reached by the authors provide a basis for further study of the integration of these two paradigms and a preliminary integrated model of the two methodologies.

INTRODUCTION

In San Jose, California, in June of 2004, the San Jose Police department began using a new mobile dispatch system in every patrol car. Police officers commented that, "the system is so complex and difficult to use that it is jeopardizing their ability to do their jobs"[6]. The police officers were not consulted about the design of the user interface and they felt the interface was unsatisfactory. Several experts, consulted by the New York Times, agreed the user interface was unsatisfactory[6]. How could this happen with an interface that must be used in emergency situations? Apparently, this was a $4.7 million off-the-shelf system from Intergraph Corporation, which was in use in other cities. It certainly makes one wonder how police officers are using, or not using, the system in these cities.

Most people believe that computers are tools to accomplish something. Computers do what they are told; they do not have an agenda. If problems arise from technology, the users are to blame[1]. It is easy to blame the users (as management did in the case of the San Jose Police Department), but in this case, as in most other cases, it appears there are other causes. Historically, for the software developer, the user interface has been much less important than the underlying software. The software development machine is focused on achieving its defined goals of efficient development and product delivery. The user is forced to adjust to the system that resulted from the programmer-centered development effort[5].

Alan Cooper[2] commented that he could not see the broader perspective of software development until he extricated himself from the programming grip. He claims that only then did he see that programming is such a difficult and absorbing task that it dominates all other considerations, including the concerns of the user. Since the early 1990s, Cooper has focused most of his efforts towards the new discipline of human computer interaction (HCI) design.

In early systems, there was no user interface; the "non-expert" user never got nearer to an early computer than a printout. Now, the graphical user interface (GUI) and the supporting interaction software are critical development issues for almost every system. Most people can cite programs that have excellent and poor user interfaces. Getting the user interface right and having it work properly with the underlying software is never easy, but by properly using the disciplines of software engineering and human computer interaction, developers can be successful more often.

Software Engineering

Software engineering focuses on large software development projects involving anywhere from tens to thousands of programmers/system engineers. A formal definition of software engineering is found in IEEE Standard 610.12-1990[8]:

> Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

Software engineering has been very successful in the development of complex software projects, but designing an excellent user interface is not an area of focus. McBreen[11] states that software engineering was invented to tackle the problems of really large NATO systems projects. Since the genesis of software engineering, McBreen emphatically asserts that the needs of the U.S. Department of Defense have dominated the conversation about software engineering.[11]

In many software development efforts the user interface is not a key issue, it is an afterthought[10,18]. It is assumed that once all the user requirements are met by the underlying programs, a user interface can be rapidly designed as part of the integration process. Sometimes this is accomplished by someone with an HCI background, but it is often accomplished by a junior software developer needing to gain experience in designing a user interface.

From the authors' experience, the dominant attitude among software developers is that the user serves at the behest of the software development team and getting the software development completed efficiently and on time is the most important goal. This has created a programmer-centered development approach[5]. The results are programs with excellent internal design that work exactly as designed, but users are totally baffled by how to use the program. To resolve this problem, computer scientists and experts in user interfaces have joined together to create the discipline of HCI design.

Human Computer Interaction Design

HCI design is concerned with the design, evaluation, and implementation of interactive computing systems for human use[5]. HCI design emphasizes the usability of the system in the development process. Since computers and their programs have become an essential part of many work places and homes, they need to work reliably and be easy to use. "Computers do not take risks. Like clerks who mindlessly go through the motions and come to ridiculous conclusions about how much change they owe their patrons, computers do not pay attention to what they do… Computers cannot step back or out of the program and judge whether their

actions make sense within a larger existential (changing) context"[9]. Software developers need to take on this task of successful integration of computers and user needs.

HCI community members have developed their own set of user-centered design techniques that focus on usability. These methods are underutilized and sometimes difficult to understand by software development teams. In addition, HCI advocates have developed their own techniques and tools for use during the software development cycle[18]. Usability techniques are expensive and time consuming, although there are approaches that decrease the cost[13]. Many software engineers recognize (at least off the record), that many of the HCI techniques are useful and appropriate for projects that have user interaction as a major component. Integration between these two disciplines is certainly possible and probably required as software development becomes a more user-centered activity.

When potential computer professionals begin training, they start by learning to program in one or more computer languages. It is not until later in their education that they learn that programming, as interesting as it is, is not the focus of computer science. After graduation, they will probably spend more of their working career doing software engineering design or testing software someone else coded[3]. An HCI course is typically an optional course, although a few computer science programs are making it mandatory. As the user interface becomes more central to applications, some kind of integration between these two disciplines becomes more important[18].

SOFTWARE ENGINEERING AND HCI IN THE CLASSROOM

Undergraduate students at Texas A&M University – Corpus Christi must complete a senior capstone course which requires the application of software engineering principles to develop a system for a real community client[4]. Almost all of these capstone projects have had significant interactive components. Some students also incorporate the optional HCI course into their academic programs. Many graduate students take the graduate HCI course as one of their electives and they also complete community based projects.

The following projects are examples of capstone projects completed in one-semester courses. All of these projects had significant interactive components[4]:

- YMCA Administrative Support System
- Catholic Charities Aid Support System
- Parks and Recreation Client and Course Tracking System
- Driscoll Children's Hospital Vendor Verification System
- Best of the Best Companies Survey System
- Head Start Meals Tracking Support System

The following are examples of Web-based development projects selected for students to complete in one-semester HCI courses. All of these projects had significant procedural components[4]:

- Blanche Moore Elementary School Library Data Management

- Computing and Mathematical Sciences Advising and Mentoring
- Corpus Christi Parks and Recreation Department
- Mission 911 Family Shelter
- Corpus Christi Business Journal
- Catholic Charities of Corpus Christi

In these real-world, but small projects, the major development phases directed by the professor included requirements gathering, analysis, design, coding, testing, implementation, and deployment to the client. One significant difference is that HCI projects incorporated low-fidelity prototyping, while capstone projects only minimally used low-fidelity prototyping. Both types of projects had a user-centered focus and incorporated usability evaluations.

The conclusion one can draw from this sample of small projects is that software engineering and HCI can, and should be integrated to produce better products for the users who are the beneficiaries of the creative efforts of developers. It is apparent that the key that makes these approaches appear to be so similar is the user-centered development focus fostered by the professor.

SOFTWARE ENGINEERING, HCI AND PROJECT TYPES

Berdayes and Murphy[1] argue that in the current development environment, computer interface applications are developed within a narrowly defined framework. The demands of the computer system take precedence over all other criteria, and the user must adjust to the needs of the computer. This is a recipe for continuing the current problems with user interfaces and the lack of usability of application systems.

Some programs have very complex interactions between the user interface and the underlying processes, while others have no user interface at all. The amount of integration needed between the software engineering team and the usability team also varies. Software development efforts can be divided into four general classes, each of which requires a different level of integration between the software engineering team and the user interface team. These lines are very fuzzy and often a program development effort will have characteristics from multiple classes, but each class of programs requires a very different level of integration.

No User Interface – Class I

There are some applications that have little or no need for a user to interact with the software. These types of programs tend to be very complex and need the full benefit that software engineering can bring. Some examples of this class of programs are operating system components or implementations of protocols, designed to have no user interface or an interface designed for experts only. Another example is the core processes of a database management system. This class also includes programs like spell checkers, where user input is required to prompt an output from the program, but the user interface is the responsibility of another program. The results of these programs greatly affect the user, but they have no direct connection to the program through an interface. For Class I programs, traditional software

engineering practices are well suited, hence there is no need for integrating software engineering and HCI.

Minimal User Interface – Class II

A second class of programs are those that have a minimal user interface. These programs may be very complex or fairly simple, with a software development/software engineering team to match. Some examples of Class II programs are setup programs or configuration programs that are run once or in very limited situations. Unfortunately, many of these types of programs have user interface problems. Most computer scientists have experience with a setup program or other wizard, where one has no idea what should be done next, and the interface and the documentation give no assistance.

For Class II programs, some integration between the software engineering team and user interface team is necessary, but it is not a core development issue. In these cases an HCI expert could be brought in fairly late in the process to design the user interface and probably would not need to be an integrated member of the development team.

Static User Interface – Class III

Programs often present a user interface that is designed for a very specific, often repetitive process. Users become very familiar with the interface and often accomplish extended data entry without ever looking at the screen. Systems that provide input/output for transaction processing are an example of Class III types of systems. The underlying programs are often updating a database or feeding another type of complex program.

The integration between the software engineers and HCI team needs to occur, but each team can work relatively independently. Gathering user requirements and the preliminary design phase need to be a joint project leading to an integration specification, where the two teams agree on how the user interface will be integrated with the underlying software. The rest of the design phase and modeling/prototyping effort can then be completed independently, with feedback between the two teams as changes occur. After the user interface has been prototyped, coded, tested, and otherwise completed, while working very closely with the user, it can then be mated to an equally well tested set of underlying programs.

User Centric Interface – Class IV

There is a large and growing class of programs where the user interface may be more important than the underlying code; these are interface centric programs. In Class IV programs, having an excellent user interface is the difference between successful and unsuccessful systems. Many of the applications that are used on a personal computer fit this category, including word processing, spreadsheets, presentation software, software development, games, calendaring, etc. Another major class of applications are those designed for the World Wide Web. Users will not revisit commercial web sites where the user interface is unacceptable. Web users are becoming savvier on what is and is not acceptable in a program interface.

For Class IV types of programs, the user interface becomes a primary factor, no matter the complexity of the underlying program. One approach to accomplish this type of software development effort is to remove the separation/specialization between software engineering and usability experts. This has the advantage of bringing the interface/usability issue to the forefront. All software development phases need to be considered through the lens of both software engineering and usability, with compromises and decisions made with the user in the forefront. One way to accomplish this is with a complete integration of the software engineering and usability teams.

Class IV programs are more closely aligned with the types of programs seen in the capstone and HCI courses as noted in the previous section. This is the domain where the integration of these two disciplines needs to focus. The remainder of this paper discusses the integration of these two disciplines with the goal of accomplishing projects that require a software engineering approach, but also have major usability and user interface requirements. The number of Class IV projects far exceeds the sum of the projects in the other three classes. Therefore, this class is a key area of concern.

INTEGRATING SOFTWARE ENGINEERING AND HCI

To provide the integrated software development team needed for user centric software development, one should be concerned with four areas. The first has to do with how software engineers are trained, the second on how to integrate the software development team during the project, the third on shared techniques between both communities, and the fourth on incorporating a human-centered focus.

Training Computer Scientists

To be part of a complex user centric development effort, computer scientists need to be educated in both software engineering and HCI. There are several articles outlining how to incorporate training in software engineering and HCI into a computer science degree program. Seffah and Andreevskaia[17] outline a set of skills needed by those that have been educated in both disciplines. Fernandez[4,5] presents a perspective on how a training program might be set up using real-world experiences in HCI. Seffah & Metzker[18] discuss why HCI training should become part of the core curriculum in computer science; they also suggest that training in both fields should be part of hiring managers' employment criteria for software engineers.

Integrating the Software Development Teams

For a truly integrated process there needs to be an integration of software engineering and HCI from start to finish. For example, the requirements gathering process needs to be accomplished by team members with expertise from both disciplines, continuing through planning, modeling, coding, testing, and deployment. Those with usability expertise will most likely focus on the user interface aspects of the project, but they need to be aware of what is happening in all parts of the project and be able to change the direction of the project to meet the needs of the user. If usability experts focus only on the user interface, it is likely that a user centric system will not result.

Seffah and Metzker[18] provide four different models for involving usability experts in software development teams: use a third party company that specializes in usability engineering, involve an external consultant in user centered design as part of the development team, form/create a separate usability group or team, or provide training to some members of the development team that will act as usability experts. These models provide a possible roadmap of how to provide usability expertise and integrate HCI into the software development process. Choosing which model is a decision based on the project and available resources.

Utilizing Shared Techniques

There is commonality between these two disciplines; many of the methods and techniques are shared. Four examples are discussed below.

Pressman's textbook, *Software Engineering: A Practitioner's Approach*[15] and Preece, Rogers, and Sharp's textbook, *Interaction Design: Beyond Human-Computer Interaction*[14], both cover Barry Boehm's spiral model, but in very different ways. Pressman[15] approaches it as a core model built for the software engineering lifecycle. Preece et al.[14] cover it as a stepping-off point, leading to the HCI life-cycle models; the difference is in focus. The HCI community focuses on the user whereas the software engineering community focuses on the project. This duality can be used as part of an integrated effort. The spiral model can be the basis of the overall software engineering effort, but when working the user interface, the more specialized HCI life-cycle model comes into play as a key part of the design effort.

Prototyping is a technique common to both disciplines. Low fidelity prototyping is normally employed in all HCI design projects. This technique could be easily incorporated within any requirements gathering and analysis process. McCracken and Wolfe[12] emphasize using poster board type materials to make the low fidelity prototypes. This works well, but any form of paper products appears to work. Low fidelity prototypes invite users to engage in the development process. This technique allows users to easily participate in the discussion of the design and to make changes of the proposed user interface.

Use cases are another shared technique. A use case is associated with an actor, someone using the system. The use case captures the set of actions that the analyst believes will be most commonly performed. Use cases can be described graphically, as a narrative, or as a list of actions to be performed[14]. Use cases can be a starting point for specifying end-user needs and integrating human factors into the development process[18]. Use cases fill an important role for both disciplines; they provide a tool to document user interactions with the proposed system. Use cases can be jointly developed by a team of software engineers and HCI experts. The two teams will have different requirements for the use case, because of their different focus. The HCI team will take the information from the use cases as a lead in to task analysis, while the software engineers will use the information from the use cases as part of requirements gathering process and as a key source document for the design phase. By jointly working the use cases, a much more integrated team and process will occur, with the overall software development process having a more user centric focus.

Work modeling is a technique used as part of usability analysis to model the work the user will be performing[14]. It is similar to the component-level design process as outlined by Pressman[15]. Either tool could be the basis for the usability analysis or modeling components. In many cases, effort would be saved by accomplishing one set of documentation that is used by the integrated team, instead of two separate efforts covering the same area by separate groups of experts.

Incorporating a Human-Centered Focus

Usability and software experts were interviewed in a study by Seffah and he found that they all agreed that generic "soft skills" were very important in the field[16]. These skills are listed as essential in many professional positions that go unfilled because of a lack of available candidates. A large international firm that regularly hires Texas A&M University – Corpus Christi computer science graduates has stated that the corporation is only interested in hiring students who have the requisite people skills to be good team players and who have a user-centered sensitivity. Seffah's survey found the most critical generic skills that emerged from the survey were primarily related to writing, presenting, communicating, and working with clients and end-users[16].

Seffah's work and research at Concordia University in Canada has led to the creation of a Human Centered Software Engineering Group[7]. This group seems to be in the forefront of bridging the gap between human centered approaches and software engineering practices. An expansion of these efforts is needed throughout the research community so that more rapid progress may be made to enhance methodologies to truly satisfy the end-users. From the classroom laboratory experiences mentioned earlier, this human-centered focus appears to be the missing ingredient in software engineering approaches.

There are other areas of commonality between the two fields that need to be explored so that true integration between the disciplines occurs. The goal of the software development process is to provide a satisfying user experience with the newly developed software. By closing the gaps between the two disciplines, there is a higher probability of successfully accomplishing this goal.

CONCLUSIONS

Developing complex computer systems is never an easy process. It becomes a more difficult process when the user interface plays a larger role. Having a complex user interface makes this already difficult process much more complicated. The traditional method has been a separate team of usability experts who focus on the user interface, with a limited impact on what occurs behind the user interface. A worst case scenario is software engineers with little or no usability training to accomplish the user interface as part of the integration phase. This approach is never ideal, but it is acceptable for software development efforts for applications where the user interface is not a key part of the application.

The equation changes as soon as the interface and user interaction with the interface become key issues, as is the case with most current software development efforts. One approach for a successful development environment for these user-centric applications is an integrated approach with software engineers and usability experts working as a team. There are several different

acceptable models that allow this to happen; there is some crossover between the two disciplines which makes the integration effort easier. The key factor is finding team members with experience in both fields, preferably with formal education in both disciplines and work experience in both roles. The golden rule in all cases is to foster a user-centric focus within all software development team members.

REFERENCES

1. Berdayes, Vincent & Murphy, John W., Computers, theorizing, and practice, in Berdayes, Vincent & Murphy, John W. (eds.), *Computers, Human Interaction, and Organizations: Critical Issues,* Praeger Publishers, Westport, CT, 2000, pp 1-13.

2. Cooper, Alan, *The Inmates are Running the Asylum,* Sams Publishing, 1999.

3. Denning, Peter J., The field of programmers myth, *Communications of the ACM*, 47, 7, July 2004, 15-20.

4. Fernandez, John D., Engaging students with community organizations by using computer technology, *SIGITE 2004 Conference,* October 2004.

5. Fernandez, John D., Human computer interaction closes the software engineering gap, *Proceedings of the 2004 American Society for Engineering Education Conference and Exposition,* 2004.

6. Hafner, Katie, Wanted by the police: a good user interface, *New York Times,* November 11, 2004. Retrieved November 11, 2004, from http://www.nytimes.com/2004/11/11/technology/circuits/11cops.html

7. Human Centered Software Engineering Group, Concordia University, Montreal, Canada, URL: http://hci.cs.concordia.ca/www/hcse/. Visited December 14, 2004.

8. IEEE, Standards Coordinating Committee of the Computer Society of the IEEE, IEEE Standard 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers, New York, NY, 1990, p. 67. Retrieved from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=4148 on December 14, 2004..

9. Kramer, Eric M., Contemptus mundi: reality as disease, in Berdayes, Vicente & Murphy, John W. (eds.), *Computers, Human Interaction, and Organizations: Critical Issues,* Praeger Publishers, Westport, CT, 2000, p. 38.

10. Mayhew, Deborah J., The usability engineering lifecycle, *CHI 99 Conference on Human Factors in Computing Systems*, CHI '99 extended abstracts on human factors in computing systems, Pittsburgh, Pennsylvania, tutorials, pp. 147 – 148. Retrieved November 27, 2004 from http://delivery.acm.org/10.1145/640000/632805/p147-mayhew.pdf?key1=632805&key2=6290061011&coll=ACM&dl=ACM&CFID=32320067&CFTOKEN=35595349

11. McBreen, Pete, *Software Craftsmanship: The New Imperative,* Addison Wesley, 2002.

12. McCracken, Daniel & Wolfe, Rosalee, *User-Centered Website Development: A Human-Computer Interaction Approach*, Pearson Education, Inc., Upper Saddle River, NJ, 2004.
13. Nielsen, Jakob, Guerrilla HCI: using discount usability engineering to penetrate the intimidation barrier. Retrieved November 20, 2004 from http://www.useit.com/papers/guerrilla_hci.html (This paper was one chapter in the book Cost-Justifying Usability, edited by Randolph G. Bias and Deborah J. Mayhew), 1994.

14. Preece, Jennifer, Rogers, Yvonne, & Sharp, Helen, *Interaction Design: Beyond Human-Computer Interaction,* John Wiley & Sons, Inc., New York, NY, 2002.

15. Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Sixth Edition, McGraw-Hill, New York, NY, 2005.

16. Seffah, Ahmed, Learning the ropes: Human –Centered design skills and patterns for software engineers' education, *Interactions*, Sep-Oct 2003, pp. 36-45.

17. Seffah, Ahmed & Andreevskaia, Alina, Empowering software engineers in human-centered design, *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, 3-10 May 2003, p 653-658. Retrieved from www.ieeexplore.ieee.org on December 13, 2004.

18. Seffah, Ahmed & Metzker, Eduard, The obstacles and myths of usability and software engineering, *Communications of the ACM*, 47, 12, December 2004, pp. 71-76.

Bibliographical Information

JOHN D. FERNANDEZ
Dr. Fernandez is Assistant Professor of Computer Science in the Department of Computing and Mathematical Sciences. Having served 20 years in the U.S. Air Force and 10 years in private industry, Dr. Fernandez brings real-world experiences into the classroom for his students. His research interests are in HCI, information assurance, and software engineering.

LARRY YOUNG
Mr. Larry Young is a retired Air Force officer who served 22 years in various technology leadership positions in the U.S. Air Force and 5 years in private industry. He is currently a computer science graduate student and an adjunct instructor in the Department of Computing and Mathematical Sciences. His current research interests include software engineering and databases.