

Multi-faceted development

Ebba Thora Hvannberg

University of Iceland, Hjardarhaga 2-6, 107 Reykjavik, Iceland

ebba@hi.is

Abstract: In this paper we present an example of how two disciplines Software Engineering and HCI can work side by side on the same problem, having the same goal but with different approaches. Such work can help us understand the similarities, differences and the synergies that multi-faceted development can bring. It should motivate us to understand the goals of the disciplines, and their role in the future development environments. Experience with working on this particular case has convinced us that it is important to continue to develop two strong disciplines but that interfaces between them must be built.

Keywords: HCI, Software Engineering, Quality, Case study, Facet

1 Introduction

Human computer interaction (HCI) is an interdisciplinary field, interacting with many areas. It is a part of computer science, but has strong roots in psychology, sociology and industrial design. “Human Computer Interaction is concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena, surrounding them” (Hewett et. al 1996). HCI includes knowledge from computer science in the area of application design and interaction design and the engineering of user interfaces. To understand what it means to engineer the user interfaces, we can relate to the definition of software engineering as defined in IEEE 610.12 (IEEE 1990) “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

The standard 13407 (ISO 13407 1999) recommends using an interdisciplinary team in user interface design. Yet, applications show there is much research needed to understand how they can interact (Bryan-Kinns and Hamilton, 2002).

More interesting ideas are often created in an interdisciplinary collaboration. Solutions to problems appear when you look at them from multiple perspectives. We should not limit ourselves using just one model to describe user interfaces or software or one view but multiple ones. We may

connect these models with interfaces so that they can be used together. All too often we have tried to build large modeling or programming languages that either encompass everything or dilute ideas that are reduced to some least common denominator.

The types of interaction between humans and computers are evolving. We have different software and hardware technologies. The problems we need to solve, the users and the physical and spatial contexts are changing. As we discuss the two areas, we need to be careful not to delve too long on the past but look at the roles and the interaction of these disciplines in the future and in relationship with other disciplines too.

In this paper we present an example of the two disciplines working side by side. We argue that it is not necessary to close the gap but to bridge it with appropriate interfaces so that cooperation between disciplines and people studying them can take place. Such case studies of multi-faceted development will help us understand the differences between HCI and SE and the synergies that the disciplines can form.

2 Unified goals

Software engineering is concerned with process and product quality. ISO 9126-1 (ISO 9126-1 2001) divides product quality into external and internal quality on one hand, and quality of use on the other. External and internal quality includes functionality, reliability, usability, efficiency, maintainability and portability. According to the standard, quality in use

is the user's view of quality and includes effectiveness, productivity, safety and satisfaction. Internal quality indicates external quality that again indicates quality in use.

Additionally, software engineering monitors the use of resources while building, maintaining and operating the system.

Standards that cover quality of HCI have similar quality goals. One can continue to specify such quality goals and I can for example name trust as being a perceived measure (Corritore, Kracher, Wiedenbeck, 2003). However, HCI has also been concerned with the effect the system has, such as its social impact (Shneidermann and Rose, 1990). There are yet other types of goals, but they are related to functionality of the system for solving the problems, such as sustainability, economy, physical and mental health, sociability, learning etc.

Although the goals of the two areas look similar, the perceived difference between the two areas may be that software designers think that the system should solve as much of *a problem* as possible, and look at the user as an operator. HCI looks at the human as a primary actor of the system that completes *a task*.

3 Network of human and system actors to solve problems

Undoubtedly, in the future there will be more automation and less human operation. This is especially apparent in areas where users are mediators between the system and primary initiators of tasks. Twenty five years ago I worked as a telephone operator in a telecommunication company. When a caller wanted to make an international call, he had to call the calling center where he was connected. There, operators wrote requests on paper forms that were given to other operators that made the call. The length of the telephone call was then recorded on -paper. Automating this bookkeeping of the telephone requests would have been solving the wrong problem. As we know, the telephone operator has become mostly obsolete. Today, we see many examples of similar human computer interaction patterns, e.g. in air traffic control, hamburger fast food restaurants, the classroom etc. The third party operator will gradually have a diminishing role and even disappear as a mediator, and the initiator of the task can directly solve his problem with the aid of a computer. People are no longer constrained in performing work in some certain environment, they are free to stay or move around.

Traditionally, software developers have looked at one type of user and one system. Users are now participants in a world assisted by diversity of computers. Development methods are beginning to acknowledge this change. Newer approaches must view a user of many systems and even many users collaborating in using those systems.

Michael Jackson (Jackson, 1995) reminds us to look at the problem domain, to analyse what problem needs to be solved, not who solves it or how. We have been taught to separate the what from the how, but there are three stages. The first stage, the problem stage, is to define what problem we need to solve, the second part, the boundary definition, is to decide who solves the problem, that is how the work is divided among people and systems, and the third part, the design including the architecture, how it should be done and how the subsystems and people interact.

We will further focus on the second stage. Before we design how to solve a problem, we need to define who should solve it. Should it be our new system, other systems, and should people solve a part of the problem even manually. Normally, the people we ask to solve a problem will do this with the help of systems. Just as we define the interfaces between our new system and other systems, we also define the interfaces between our new system and the users that are solving a part of the problem cognitively. We even may go as far to define the interfaces between humans, i.e. human to human interfaces. This may very well influence the human to computer interaction. Since it is a network, we want to find out to how all the components communicate. All that may affect the performance of the solution. As in a network, a single component can slow down the end-to-end performance.

In the next section we will describe the role of people in our networked system as has been described above.

4 The role of humans in an evolving world view

4.1 Humans as resources

In the beginning of this paper, we described HCI as being a part of computer science. We can look at HCI in a similar way as we treat databases, networks, and operating systems. Those are different disciplines that help us design and implement certain parts of the system – interaction design is comparable with these areas. HCI is broader since it covers also the second stage, i.e. Boundary

specification, as is described in the previous section. Once the boundary specification is in place we can carry on with Architecture specification where we decide who does what and how they interact. Note however that HCI is not as broad as to cover the problem definition, i.e. the first stage because at that stage we have not decided who does what. Some methodologies define the interaction between human and computer prematurely, before having decided the distribution of the workload.

The human is a resource, similar to the physical network, CPU or disk space. They are constraints just as disk access or bandwidth. However, requirements specification usually doesn't put cognitive capabilities forward as a measurable entity or formal constraint. What makes it of course harder to design for is its variability. Unlike physical resource, a human resource does not have a guaranteed capacity but can be increase through learning or decreased under external influences.

4.2 Humans as natural systems

Computer systems receive information from mechanical and electrical systems and provide feedback to them, e.g. airplanes or fish processing assemblies. Computer systems also receive such input from natural systems, e.g. earthquakes, indirectly through sensors and meters. In much the same way, people are sources of data and action and receivers of input.

We thus need a model of the human to understand how it interacts with the computer system, similar to other natural systems that a computer system interacts with. Computer systems capture information from natural systems and try to store information about them and even predict their behaviour.

4.3 Resource capabilities and constraints

When determining who should solve the problem, we need to know what resources we have. How can the resources meet the required achievements? How performing are the computers, the network, and the people. Do all these together meet our performance expectations for the problem solving or do we need to boost their performance. Additional resources e.g. of memory and processing power can be added to a computer, networks can be scaled and people can be trained to increase their performance. The software that we are building can also help meet the gap by designing it to compensate for the lack of human resources. Operating systems can manage resources better; network services can balance the load to

computers. Functionality can be added to systems to increase people's capabilities, increase their perception, processing and projections. Caches do the same for client systems in distributed systems; they enhance the performance of clients by giving fast access to data or functions of servers. As with network bandwidth, interaction between user and system can be enhanced.

On one hand we define the quality of the solution, e.g. its expected performance, our confidence in the solution (security, reliability, usability), and its adaptability to change (maintainability, portability etc.). On the other we are constrained by the computational and network resources, organizational (including cost to produce the system), regulatory, physical and social (including ethical) contexts as well as the human resources to solve the problem as participants in the network and their cognitive capabilities.

5 Different approaches – an example from air traffic control

This section describes two different approaches of people addressing a problem of integrating the user interfaces of three different systems of an air traffic control workstation. The systems that should be integrated are electronic flight strips, radar and communication system. Although I talk about software engineers and HCI engineers I should warn that the other might not exclusively in either group but somewhat influence people.

When software engineers design a system, they will first try to describe the problem or the tasks by visiting its most abstract form, almost naively. Then as more knowledge is gained about the problem and the interfacing environment, they will refine the description with more detail, using generalization and aggregation hierarchies. This is the approach we used when given the tasks of integrating different systems, like radar, communication and electronic strips in the air traffic controller's workstation. Software engineers would then take the description and verify it by asking the users or observing them at work. We described entities, behaviour, relationships between entities, communication between behaviour, events, actions as responses to events, users and other stakeholders and the environment. SE described these as text or diagrams. The definition was done in several stages, with the most abstract one first, followed by a refined one including the technologies that are used for air traffic control. The problem with this approach is that we users are not very good at interpreting

abstract knowledge without seeing any details. Yet, it may be the only way to manage complexity.

People with HCI background took a different approach. They went directly to site visits, prepared with questions that would answer what the constraining factors were. These are common approaches in the HCI area such as Contextual design (Beyer and Holzblatt 1998) and Cognitive Work Analysis (Vicente 1999). The context was primarily viewed by understanding how controller gained awareness of the situation (perceive, gather, project – (Endsley 1995)) followed by a plan of solution and an action (from the Norman Action cycle (Norman 1986)). The approach also included drawing the flow of data or the relationship between system components. This approach is more bottom up, and tends to look at details and then to classify and analyse the work.

Prototypes, and scenarios are also detailed forms that give users a concrete model to look at and judge. Reviewing methods such as heuristics analysis looks also at details.

A secondary goal of this study is to understand how we can benefit from looking at a problem from different sides. Our next task will be to see whether these two approaches, i.e. their models cannot be bridged with two interfaces. We should not necessarily try to find a common denominator but build interfaces that will allow the two research areas to cooperate.

Psychologists are looking at detail, but software engineers at abstraction. There doesn't always seem to be so much a difference between SE and HCI, but when an SE looks at a problem, he tries to abstract in the beginning, classify entities and define abstract interfaces to entities. Then as the development proceeds refinement occurs that adds more detail and implementation. Anthropologists and psychologists prefer to look at details, and learn how humans interact with systems or solve problems. This tends to cause conflicts between the two disciplines. (Bentley et. al 1992) Similar discussion (Diaper 2002) has taken place when debating how appropriate scenarios are for user interface design. Whereas scenarios may seem to be detailed and concrete, other task models are often viewed as more abstract. The conflict rises because we want to show users concrete examples because we don't trust them to be able to think abstractly about problems. On the other hand we want to build abstract task models that free us from deciding on the interaction design. We also want to use more abstract models, and try to stimulate ideas of future use. In our work when designing interfaces for a

workstation that has new and heterogeneous technological facilities, we have found that abstract models that hide e.g. technical and organizational detail become durable.

To understand one another we need to understand what is the different in models originating from the HCI and SE areas. There are user interface models, cognitive models and system models. Is there a difference in what they show e.g. functionality or non-functionality attributes such as performance and fault tolerance and how, e.g. abstraction.

6 Learning opportunities

The two areas have used different approaches to development but they have also influenced one another. One can see that action research originated from social sciences, soft systems methodology and participative design has had influence on software life cycle models. Software engineering has been rather late at realizing the close cooperation with clients. It is only now with XP (Beck 1995) and Evolve that close cooperation with the customer is being formalized into a framework. Iterative approaches are gaining popularity. Empirical research studies and benchmarks have been performed in the area of HCI and software engineering needs to make intelligent decisions based on evidence. Claims analysis has been used in scenario based design for user interfaces (Rosson and Carroll et al 2001) and scenarios are also used in architecture trade off assessments (Clements, Kazman, Klein 1995).

There are certainly many areas where the two disciplines can learn from one another. HCI can look at how software engineers have tested systems and built architectures. Software engineers can listen better to stakeholders, not just for functional requirements but also for technical, social and economical constraints under which a system is built, maintained and operated. Not only at the onset but also throughout the lifetime of the system.

7 Building interfaces between disciplines

Our case study has convinced us that we need not find a common denominator but build interfaces between disciplines that will allow the two research areas to cooperate. It may also be of benefit to try to make the distinction between the two disciplines clearer, e.g. so that both are not trying to specify requirements. As we have seen in papers of previous

HCI-SE workshops (Willshire, 2003 and Clemmensen and Nörberg, 2003), it becomes very difficult to merge two fields in a curriculum because the combined course will suffer from lack of time. Instead, we should continue to build strong disciplines of HCI and Software Engineering and they can learn many things from each other but also from other disciplines and apply it to their own areas but alas to a different problem. We also need to build good interfaces between HCI and Software Engineering so that they can exchange views. Not only should the interfaces be built so that they can divide the work but also so that there can be multi-faceted development, each taking their own road to their destination and hopefully benefiting from the multiple facets, either by extending the understanding or via the choice of one facet over the other.

To motivate the discussion on what these interfaces are, I give several examples such as Abstract and Concrete, Intuitive and Reasoning, Aesthetic vs. Heuristics, Human and System, Empirical and Action research, Models vs. Prototypes. Although these are pairs to show that you can apply both techniques, these are not set forward to put one of the items in the pair into the SE category and HCI in another. The objective is simply to show that one can benefit from developing multiple facets.

8 Conclusion

In this paper we have discussed the two disciplines HCI and SE. We have shown their relationship with one another and pointed out their unified goals in quality. The need for considering the changing role of human in future system architectures was also discussed. We presented a brief description of two different approaches to building an air traffic control workstation. The comparison of the two approaches will be a part of future work. The case study is described here to motivate more such multi-faceted development that hopefully will have meaningful common interfaces on the facets' boundaries.

References

- Beck, Kent, *extreme Programming Explained*, Addison Wesley, 1995
- Bentley, Hughes, Randall, Rodden, Sawyer, Shapior, Sommerville (1992), Ethnographically-informed systems design for air traffic control, *Proceedings CSCW 1992*
- Beyer, Holtzblatt, (1998) *Contextual Design*, Morgan Kaufman
- Brian-Kinns, N & Hamilton, F. (2002) One for all and all for one? Case studies of using prototypes in commercial projects. *NordiCHI: Proceedings of the Second Nordic Conference on Human-Computer Interaction, October 19-23, 2002, Aarhus, Denmark*, p. 91-100
- Clements, Kazman, Klein (1995), *Evaluating Software Architectures*, Addison Wesley,
- Clemmensen, Nörberg (2003), Separation in Theory – Coordination in Practice, *SE-HCI Workshop at ICSE, 2003* <http://www.se-hci.org/>
- Corritore, Cynthia L. Beverly Kracher, Susan Wiedenbeck, On-line trust: concepts, evolving themes, a model (2003), *International Journal of Human-Computer Studies* 58 737-758
- Diaper (2002) Task scenarios and thought *Interacting with computers*, 14 (2002) 629-638
- Endsley, Mica R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1), 32-64.
- Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank (1996) ACM SIGCHI
- IEEE STD 610.12-1990 (1990) *IEEE Standard Glossary of Software Engineering Terminology*
- ISO 13407:1999 (1999) Human-centred design processes for interactive systems
- ISO 9126-1 (2001) Software engineering – Product quality – Part 1: Quality model
- Jackson, M. (1995) *Software Requirements & Specification*, Addison Wesley, ACM Press
- Norman D A, (1986) Cognitive engineering. *In User centred system design, D A Norman and S W Draper Eds* Erlbaum Hillside, NJ 31-62
- Rosson, Carroll et al (2001) Scenario-Based Development of Human Computer Interaction, Morgan Kaufman
- Shneiderman, B., A. Rose (1996), "Social Impact Statements: Engaging Public Participation in Information Technology Design," in C. Huff, ed., *Computers and the Quality of Life: The Proceedings of the Symposium on Computers and the Quality of Life*, ACM Press, New York, 1996, pp. 90-96.

Harning & Vanderdonckt

Vicente, Kim J. (1999) *Cognitive Work Analysis*,
Lawrence Erlbaum associates,

Willshire, M. J. (2003) Where SE and HCI Meet: A
Position Paper, *SE-HCI Workshop at ICSE, 2003*
<http://www.se-hci.org/>