The modelling gap between software engineering and human-computer interaction

José Creissac Campos
Departamento de Informática
Universidade do Minho, Campus de Gualtar
4710-057 Braga, Portugal.
Jose.Campos@di.uminho.pt

Abstract

The theories and practices of software engineering and of human-computer interaction have, to a great extent, evolved separately. It seems obvious that the development of an interactive system would benefit from input from both disciplines. In practice, however, the communication between the two communities has been difficult. Models can be a particularly good tool for communication. For that to happen the differences between the models used by each community must first be identified and understood. This paper looks at the gaps between the models used by the software engineering and the human-computer interaction communities. It identifies where differences between these models can be found, and some aspects that need addressing in order to promote better communication.

1 Introduction

Developing quality interactive systems is still a difficult and complex endeavour.

One of the leading banking institutions in Portugal has recently released the latest version of its e-banking site announcing a much richer set of features available to costumers. One particular feature that was implemented was a search mechanism over transactions which allows costumers to search according to a very flexible set of criteria. Unfortunately the simple task of checking the last few transaction of an account has been complicated. The users must now always go through the more powerful (but more complex) search interface. Any reasonable usability analysis would most probably suggest maintaining some sort of quick access to the latest transactions, keeping the power search as an option to the user.

One well known car manufacturer has spent a considerable amount developing a completely new version of its middle range car. Despite all effort put into the car's development, the air conditioning controls have obvious usability problems. This happens due to two main reasons: lack of appropriate feedback on the system's state in its user interface (the user interface presents the automatic mode as having two possible states — on or off — when in fact it has two additional semi-automatic modes); difficulties in undoing some actions (it suffices to press a button to turn the system on in the special mode to clear windscreen condensation; pressing the button again, however, switches off the condensation clearing mode only, leaving the system functioning on the previous mode used — this also means that the effect of pressing the button becomes unpredictable under these circumstances). These are obvious usability pitfalls that seem to have gone unnoticed during design and development of the system.

For the two systems just mentioned costumers' perception of quality is particularly relevant. Nevertheless, they both suffer from obvious usability problems. This shows that there is a way to go in better integrating usability issues into systems development.

This paper stems from work that is under way towards an approach to incorporate usability issues into software development from the very early stages of design (see [3, 4]). The applicability of automated reasoning tools to the identification of potential usability problems has, in particular, been studied [4]. In this context we have been in contact with both software engineering and human-computer interaction practitioners/scientists. The difficulties in communication between the two communities have quickly become apparent.

In our proposal communication between software engineers and HCI experts is based around models. This

paper reflects on the gaps between the two communities and on approaches to reduce them. In particular, in the gaps that can be identified when it comes to the use of models to design and reason about systems.

2 Software Engineering vs. Interactive Systems development

The theories and practices of software engineering and those of human-computer interaction (HCI) have, to a great extent, evolved separately. Software engineering deals with the construction of software systems. Despite its infancy, from programming technology to software development process, a large body of tools and knowledge has been produced (cf. [9]). However, developing software is still a mostly difficult and complex process. A study by Eason, cited in [18], concluded that only 20% of deployed systems were considered successful, while 40% were actually rejected. Other studies have shown that one third of software development projects is abandoned before completion.

HCI is concerned with the process of communication between humans and computer systems. For the purpose of this discussion we will consider the case of software interactive systems. This field is younger than software engineering , but also in this case a considerable body of knowledge has been developed (see, for example, [6, 14]).

Since the focus is on the interaction between system and users, the techniques developed within HCI for interactive systems development deal mainly with what can be called the interface layer of systems. Despite all progress, it is also true for interactive systems that developing them is a difficult and complex process. It is estimated that 60% to 90% of all system failures can be attributed to problems in the interaction between the systems and their users [8]. The problems with interactive systems development are not particularly surprising since interactive systems are a special case of software systems with the added complexity of having to cope with human activities, goals, capabilities and limitations.

Practitioners from both fields have developed distinct skills. It seems obvious that the development of an interactive system will benefit from the input from both fields of knowledge. Decisions regarding the user interface design can have serious implications on the implementation of the whole system, not just the user interface implementation. Even in the case of non-interactive systems, input from the HCI community can help in understanding the impact of the system in the overall context when this context involves humans.

In practice, however, the cooperation between the two communities has been difficult. This can be attributed to a number of factors. Two such factors are:

- different views on where the development focus lies — software engineers are mainly interested in solving the technical difficulties faced when implementing a given functionality, HCI practitioners are mainly interested in solving the problem of which functionality should be provided and how to optimise the way in which it is provided to users;
- communication difficulties — not always the same terms are used to describe the same concepts in the two communities; this hinders communication, at best can make it difficult and at worst can mislead the two parties into thinking that they are talking about the same thing when in fact they are not.

3 Software engineering methods vs. interactive systems development methods

In order to understand why the differences above exist we can look at typical development processes used by each community. This enables us to identify how the above identified two factors manifest themselves.

3.1 Development focus

Doing this we find that software engineering methods (for example, the Unified Process [11]) are mostly concerned with building the system. Typically the requirements gathering phase attempts to determine the functionality the system should provide and the focus quickly shifts to the issue of how to better implement that functionality. The main concerns are the quality and maintainability of the code produced. Usability issues are seldomly mentioned, if at all.

Interactive systems development methods (for example, human-centered design [10]) on the contrary, are more concerned with the design of the interaction between the system and its users. The focus of attention in on how best to support the users in performing specific activities with the aid of the system in concrete contexts of usage.

3.2 Communication

In [12] a brief comparison between the ISO 13407 standard for human-centred design (HCD) [10] and the Rational Unified Process (RUP) [11] is made. Both methods are based on prototyping and iteration. However, these terms do not necessarily mean exactly the

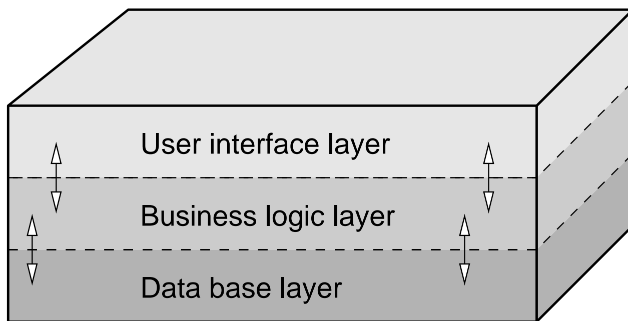

Figure 1. 3-tier architecture

same in both contexts. In RUP, prototypes are executable code and mostly seen as intermediate steps in the development of the final system (a partially implemented system). Iterations are then steps in the direction of a more deeper coverage of the requirements for the system.

In HCD prototypes are simulations or models of the user interface of the system. They are developed for usability testing purposes, and need not necessarily be executable. In this context, iterations are seen as producing new/refined interface designs, in response to the usability testing results. In fact, in HCD there is no explicit mention of producing the final system [12].

3.3 Different perspectives on development

From the above we can conclude that the two disciplines have different perspectives on development. HCI is primarily interested in developing the "*outside"* of the system. That is, the interaction of the system with its users. Software engineering is primarily interested in the "*inside*" of the system. That is, how the system is actually implemented. We will say that HCI has a black box view of the system, while software engineering has a white box view of the same system.

Software engineers think of the system in terms of its architecture. It is common for this architectural view to be organised in a succession of layers. An example of this is the 3-tier architecture presented in figure 1. Note that in this architectural model there is no mention of the user. In fact, despite mentioning the user interface, most of the initial development effort usually goes into the business and data layers, and concerns about the user interface, when present, are geared towards its implementation.

A usability practitioner will typically talk of users' goals, tasks, and user interface designs, without deeper consideration of the architectural issues and tradeoffs "*behind the scene*". The focus is on the interaction between user and artifact, not on the artifact by itself. It is usual to see references to the "interactive system" as the composition of human + system (system, in a software engineering sense).

These different views of the system (development) lead to differences on how the different available *tools* are applied. Different approaches to the use of prototypes, and to the notion of iteration step during development have already been discussed. Another relevant issue is the differences in the use of models.

4 Model based analysis and development

The use of models has become a standard technique when dealing with complexity. Models have two main purposes:

- helping understand a complex problem/solution — a good model represents a adequately simplified version of the problem/solution, making it easier to grasp what is essential about it;
- helping in communicating complex problems/solutions — once the model is produced it can be used to communicate information to others (assuming they will be able to understand it).

We can see a prototype as a special kind of model that can be executed or in some way used to simulate the system.

The term model is used for many different artifacts at different levels of abstraction. Models can vary in:

- formality — they can range from very informal "*back of the envelope*" sketches to very formal mathematical models of specific aspects of the system; typically, as the level of detail increases, so does decrease the range of features that can be expressed in the model;
- view — different models will address different aspects of the system, the UML [2] modelling language alone identifies 12 different diagram types; a typical distinction is between structural and behavioural models, in this case, however, it will also be useful to distinguish between models that take a black box view of the system, and models that take a white box view of the system;
- purpose — different needs will typically demand different types of models; a common distinction is that made between conceptual models (used for describing the problem domain), specification models (use for describing what the solution to the

problem is), and implementation models (used for describing how the solution is implemented).

In any case a process of abstraction is used to focus the attention on the relevant issues that must be considered. One of the consequences of the abstraction process is that models will reflect a partial view of the system. This view is determined by the combination of the factors just described.

5 Gaps in the modelling

Having developed (more or less) independently, and with different needs in mind, software engineering and HCI have developed different styles of modelling.

Models are a means of communication, so we can try to analyse to each extent the models in each community can serve the purpose of helping the other community.

For the purpose of this discussion we will consider that models from both disciplines can be divided into two broad categories: architectural models and behavioural models.

5.1 HCI models

In the present context we use the term architectural model to refer to those models that define the representation side of the interface. That is, how the information is presented to the user (the View in the MVC architecture). This make take the form of paper mockups of the envisaged interface, or might be developed resorting to current day IDE tools. Typically the models are developed with a specific type of interface in mind. This causes problems when the interface might be deployed in different platforms, using different interaction paradigms.

Behavioural HCI models typically define the tasks the system is supposed to support. These models capture how the system is to be used. It is well known, however, that once developed systems tend to be used in unforeseen ways and/or for unforeseen purposes. These models help shape how the interface should behave, but are of little help when considering how to implement it.

Models from the HCI community define what the (user interface of the) systems should be, but give little guidance on how to build them. This should be expected since their main purpose is helping the analysis of the systems' usability, not the analysis of their implementation. Unfortunately it creates problems when discussing the system with software engineers since they will be thinking in terms of implementation.

Interestingly this can also be a problem for the HCI practitioner. In fact, the models describe the design of the system but little knowledge/information is present regarding its usability. That type of knowledge resides mostly in the head of the usability experts that perform the analysis. Hence the models do not actually capture all the relevant knowledge about usability issues.

An exception to this are the approaches that develop user models in an attempt to capture how a user would behave in front of the system. Of particular interest here are syndectic approaches [7] that develop a model of both the system and the user. In theory they should enable the characterisation of usability concepts directly in the model. It is not clear whether developing adequate user models for this is feasible. In [1], for example, only a very simple form of behaviour (rational behavior) is considered in order to make the approach feasible.

5.2 Software engineering models

Models from the software engineering community capture how the system is built. Architectural models capture how the system can be constructed from different components. Behavioural models capture the interaction between those components in order to implement the intended functionality.

Unfortunately it is rare to have software engineering type models for the user interface layer of the system. Current day IDE tools help maintaining this problem since they are misleading. Apparently they help model the user interface, and establish a link between the two types of models. A mockup of the user interface can be used to visually design the interface, and the application code is automatically generated. Unfortunately, these tools cover a small fraction of what is needed. Graphical layout can be described, but little support is given to describing the behavioural part. On top of that, they are developed at a level that is close to the actual code, lacking abstraction.

When models are developed, they represent how the system should be built. This makes it possible to reason about the quality of the implementation. For example, whether all functionality is accessible, or if pressing some specific button causes some specific functionality to be executed. However, establishing a link between such type of models and more generic usability properties is usually not easy.

This becomes a problem, for example, when attempting to perform reverse engineering of interactive systems to reason about usability issues. The models that are produced from the code are not adequate for that style of reasoning.

Abstraction is needed to get away from actual details of implementation and into the concepts that are

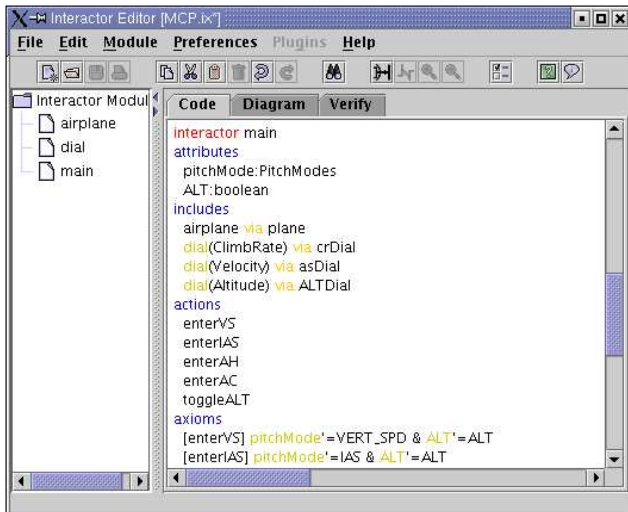

Figure 2. User interface (HCI architectural view)

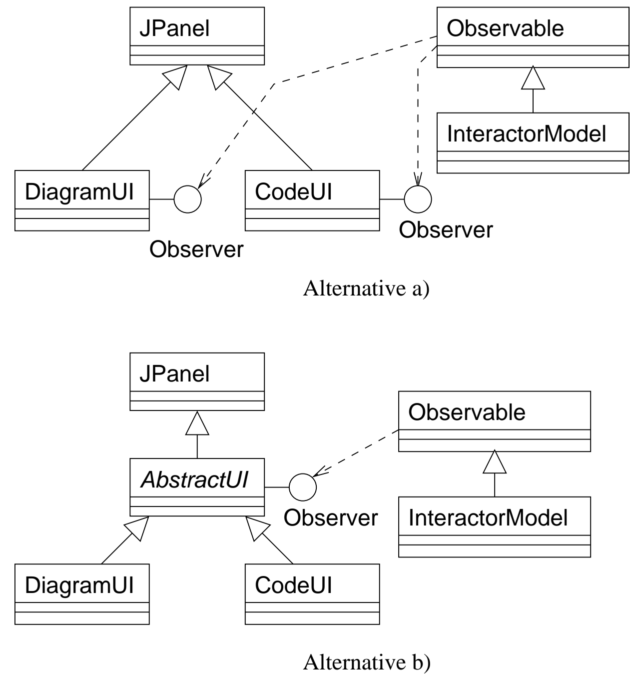

Figure 3. Observer-Observable pattern (software engineering architectural view)

relevant for usability. While the structural view of a user interface is easily apparent when using a IDE to model a user interface, the behavioural view is usually lost in a tangle of interdependent listeners objects. Hence, software engineering models are of little help when reasoning about the usability of a system.

5.3 An example

Consider, for example, the user interface in figure 2. What type of models would we use for both HCI and software engineering activities?

From an HCI perspective we would be interested in analysing the usability of the system. One technique we could use is heuristic evaluation [15]. In order to do this we would need a model describing the user interface of the system. As stated before, that might simply be a paper mockup of the intended design. The actual image on figure 2 could be part of that model.

From a software engineering point of view we would be concerned with how to best implement the system. To this end we would resort to architectural and/or behavioural models of the code implementing the system. For example, we could represent different applications of the Observer-Observable pattern to achieve user interface code/application code separation using the models in figure 3. Using these models we could discuss the relative merits of each approach.

Note that the models in figure 3 are already architectural models with strong emphasis on the user interface. A typical software engineering model would probably be more concerned with specific details of the InteractorModel class.

In the case of the HCI model, details of information representation at the user interface will be included. In the case of the software engineering model the focus is on the *internal* architecture of the system.

If we wanted to think in terms of behaviour different types of models would also be used. From a HCI perspective we could develop task models depicting how the system is supposed to be used. Figure 4 presents an excerpt of one such model in the ConcurTaskTrees notation [13].

From an software engineering perspective we could focus on the communications between the different objects in the system using, for example, sequence diagrams. Figure 5 presents a sequence diagram showing how objects communicate in the Observer-Observable pattern.

5.4 Software engineering models vs. HCI models

Figure 6 sums up the discussion above. Software engineering models are usually white box models biased towards implementation. These are models that ultimately describe how a system is coded. The specification level is usually point out as the most appropri-

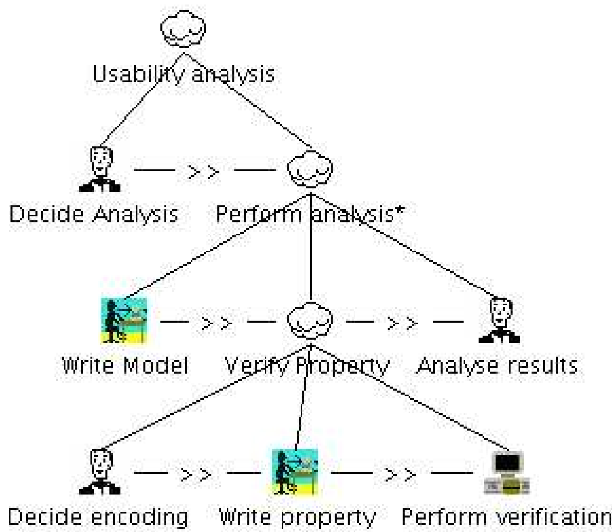

Figure 4. Task model (HCI behavioural view)

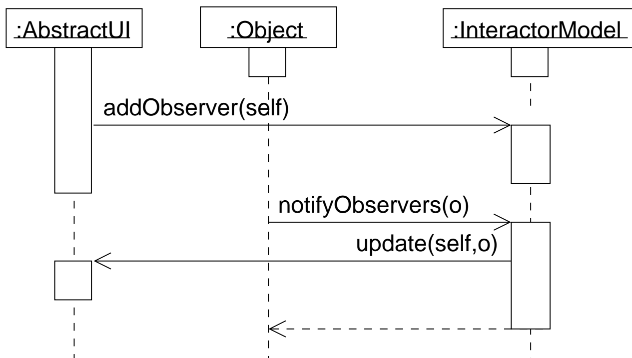

Figure 5. Sequence diagram (software engineering behavioural view)

ate level to use, but experience shows that software engineers tend to lean towards implementation very quickly.

One example of a software engineering black box model is the Use Case diagram of UML [2]. These diagrams are used to capture all the functionality that the system should implement, and are not developed from user interface perspective.

HCI models are usually conceptual models that provide a black box view of the system. They are typically used to support analysis of the usability of the system that is being designed. Some models will describe how the system will look and/or feel to the user. This means that information about the usability of the system is not explicitly modeled. This knowledge must

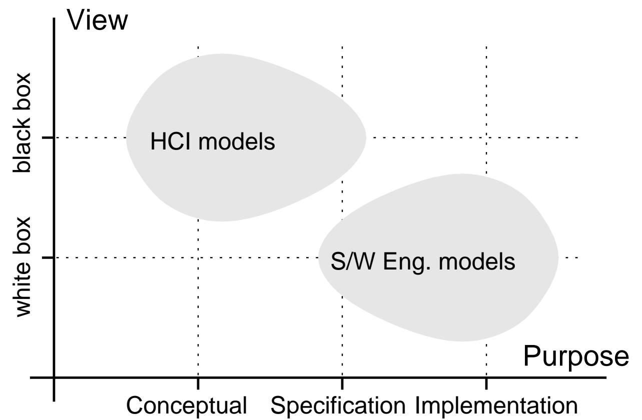

Figure 6. HCI models vs. Software engineering models

be brought to bear on the analysis either by resorting to an expert that will look at the model and analyse its features, or encoded in some specific analysis technique. Even in the last case the input from a usability expert is typically needed. It can be said that these models don't follow the usability guideline of placing knowledge in the world, instead of in the users' head.

Another approach is to take a broader perspective and attempt to model not only the system but also the user. In this case usability related information can be captured directly in the model. These models should be able to capture salient features of user behaviour when faced with a user interface, but developing them is a difficult and complex endeavor.

6 Bridging the gap

It is clear that models can be a invaluable reasoning and communication tool during software development. This is also true for interactive systems development. More so since two communities with largely different background are (should be?) involved in the process of designing and implementing an interactive system.

Architectural models are those where it will be easiest to reach common ground. At this level, HCI models attempt to capture (with varying degrees of abstraction) how the interface is structured, and what information it presents to the user. These models can be used to drive the implementation of the system. Current day IDE tools already enable the graphical definition of the user interface, with code being automatically generated. It must be noted, however, that the user interface can have implications on the architecture of the system that go beyond the architecture of

the user interface layer.

Behavioural models are considerably more far apart. This happens because models from both communities address slightly different issues. HCI models are concerned with the joint behaviour of system and users. Software engineering models are more concerned with the internal behaviour of the components of the system. Deriving one from the other might not always be an easy task. In most situations the link between the two types of modes will not be as direct as in the case of the architectural models. Behavioural models from software engineering are dependent on the architecture of the system. Additionally, typical software engineering notations do not support the kind of modelling done at the HCI level.

From a methodological point of view, since the system is developed for its users, user interface design should drive the initial stages of development. Once a first idea of the desired user interface is achieved, the process of designing a system that implements it can be started.

It is not realistic to think of software development as a sequential process. Development of software happens iteratively and incrementally. Also, the assurance of quality most be faced from the very early stages of design. Hence, usability analysis should be applied not only during the initial stages of development, but throughout all of the development process, whenever user interface issues become under consideration.

For that to happen two points need to be addressed:

- software engineering models must better cover the user interface layer — in order to promote communication the first step is that both communities are talking about the same concepts (even if in different languages).
- The relation between software engineering models of the user interface, and HCI models must be further investigated — software engineering models are developed with implementation in mind, HCI models are developed with usability in mind; nevertheless, they influence each other.

The ideas above are already being explored. In [17] the integration of a task modelling language into the UML is discussed. This seems to imply a broadening of application of the UML. This type of effort could potentially help in establishing the relation between the two types of models.

In [16] a different approach is attempted. In this case the application of the UML to the modelling and development of interactive systems is explored with the creation of appropriate stereotypes. The approach can then be seen as a step in the direction of better coverage of the user interface layer from the software engineering community. A similar approach is proposed in [5].

The definition of usability patterns could help bridge the communication gap by crystallising knowledge and best practices. However, the context of application of these patterns is complex and they can hinder the creative potential that is so relevant in user interface design. Their definition and use must therefore be made with careful consideration of all the issues involved.

7 Conclusions

Models are a particularly good tool for communication. The need for promoting the communication between the HCI and software engineering communities has been identified. For that to happen the differences between the models used by each community must first be identified and understood.

In this paper the processes and models used by the software engineering community and HCI communities have been briefly looked at. Some of the differences between the two types of models have been identified. As a result of the exercise some proposals have been made for aspects that need to be addressed in order to promote better communication between the two communities.

The end result of a better integration between software engineering and HCI will be better quality interactive systems. The notion of quality must be understood both from a HCI perspective (better usability) and from a software engineering perspective (better code).

This increase in quality should happen not only in terms of the end result (the system produced), but also in terms of the development process. That is, development should be more efficient with less need for changes to be made due to usability problems.

Acknowledgements

The author wishes to thank Gavin Doherty for his useful comments on previous versions of this paper.

References

[1] Ann Blandford, Richard Butterworth, and Paul Curzon. Models of interactive systems: a case study on programmable user modelling. *International Journal of Human-Computer Studies*, 60(2):149–200, February 2004.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Object

Technology Series. Addison-Wesley, Reading, MA, 1998.

[3] José C. Campos. *Automated Deduction and Usability Reasoning*. DPhil thesis, Department of Computer Science, University of York, 1999. Available as Technical Report YCST 2000/9.

[4] José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3-4):275–310, August 2001.

[5] Paulo Pinheiro da Silva. *Object Modelling of Interactive Systems: the UML*i *approach*. PhD thesis, Department of Computer Science, University of Manchester, 2002.

[6] Alan Dix, Janet Finlay, Gregory D. Abowd, and Russel Beale. *Human-Computer Interaction*. Pearson Education Ltd., third edition edition, 2004.

[7] D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–393, 1998.

[8] E. Hollnagel. *Human reliability analysis: context and control*. Academic Press, London, 1993.

[9] IEEE Computer Society, Los Alamitos, California. *Guide to the Software Engineering Body of Knowledge: Trial Version*, May 2001.

[10] International Organization for Standardization. *ISO standard 13407 – Human-centered design processes for interactive systems*, first edition edition, June 1999.

[11] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Object Technology Series. Addison-Wesley, Reading, MA, 1999.

[12] Bonnie E. John, Len Bass, and Rob J. Adams. Communication across the hci/se divide: Iso 13407 and the rational unified process®. In *Proceedings of the 10th International Conference on Human Computer Interaction*, Crete, Greece, June 2003.

[13] Giulio Mori, Fabio Paternò, and Carmen Santoro. Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(9), 2002.

[14] William M. Newman and Michael G. Lamming. *Interactive System Design*. Addison-Wesley, 1995.

[15] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *CHI '90 Proceedings*, pages 249–256, New York, April 1990. ACM Press.

[16] Nuno Nunes and João Falcão e Cunha. Towards a UML profile for user interface development: the Wisdom approach. In *Proceedings of UML 2000*. Springer-Verlag, 2000.

[17] Fabio Paternò. ConcurTaskTrees and UML: how to marry them? Position paper to the Tupis 2000 Workshop at UML 2000, 2000.

[18] J. Preece et al. *Human-Computer Interaction*. Addison-Wesley, 1994.