From HCI to Software Engineering and back

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

Michael D. Harrison
Department of Computer Science
The University of York, Heslington
York YO10 5DD, UK.
Michael.Harrison@cs.york.ac.uk

Abstract

Methods to assess and ensure system usability are becoming increasingly important as market edge becomes less dependent on function and more dependent on ease of use, and as recognition increases that a user's failure to understand how an automated system works may jeapordise its safety. While ultimately only deployment of a system will prove its usability, a number of approaches to early analysis have been proposed that provide some ability to predict the usability and human-error proneness of the fielded system. The majority of these approaches are designed to be used by human factors specialists, require specific expertise that does not fall within the domain of software engineering and fall outside standard software development life cycles.

However, amongst this number, some rigorous mathematical methods have been proposed as solutions to the more general problem of ensuring quality of system designs but with limited success. This paper discusses their limitations both in terms of the broader software engineering agenda and in terms of their effectiveness for usability analysis, the opportunities that they offer and discusses what might be done to make them more acceptable and effective. The paper positions those methods that have been effective against less formal usability analysis methods.

1 Introduction

Although much emphasis has been placed on engineering more reliable hardware and software systems, relatively little significance has been attached to human factors issues. In fact human factors account for a very large proportion of failures in systems and the proportion is growing as methods of software development improve. "Although valid figures are difficult to obtain there seems to be general agreement to attribute somewhere in the range of 60-90% of all system failures to erroneous human actions" [7]. A study of 1100 computer related fatalities between 1979 and 1992 estimates that 92% of these fatalities could be attributed to human computer interaction [11]. These numbers clearly indicate the need for a better integration of human-factors concerns into the software engineering life-cycle.

The use of rigorous, mathematical, methods has been proposed as a solution to the problem of improving the quality of system designs. The potential of these methods has led to their application to interactive systems development (see, for example, [6]). In recent years a focus of attention has been the possibility of performing rigorous and systematic reasoning in order to assess the usability characteristics of systems. The application of automated reasoning techniques, in particular, has been studied [15, 2, 17].

In considering the specification of a device, usability engineering is concerned with how it interacts with the context in which it is placed. The focus is not so much on the device and its structure (though this is of interest) as the way that a device is embedded in its context, that is its work environment. Therefore, as well as the specification of the device, the specification of an interactive system where the device is located is also of interest. This interactive system will involve humans and devices. It is the interplay between them that should be analysed.

Usability analysis is a difficult process, and not necessarily one where it is obvious that mathematical methods and tools can play a relevant part. Roles for the different components involved may vary and the objectives which the system is intended to achieve will also change over time. Specifying how people actually behave rather than how they should behave is extremely difficult. In this paper formal approaches to the representation and analysis of interactive systems shall be considered that aid the process of usability engineering. In particular, these formal techniques shall be compared with a class of more informal but systematic usability analysis methods.

The paper discusses the role that model based techniques can play in modelling and analysing interactive devices, assumptions about their use, and the work that they do. In the next section less formal techniques are discussed before

identifying their ingredients in the context of a more formal analysis.

2 Approaches to usability analysis

Ultimately, the only reliable method of assessing systems is to observe them in a broad enough set of situations that are as close as possible to those that are envisaged for its use. The problem with empirical analysis is that of the effort involved in constructing a prototype that is sufficiently robust to be fielded. Changes are therefore more expensive to make than they would be at earlier design stages. It is also difficult to obtain situations that are truly representative of the kind of environment in which the finished system will be used and are exhaustive of these possible situations. On the whole, successful systems evolve over time, using experiments with prototypes through trial and error.

The role of analytic techniques is not to solve all the usability problems, that would be impossible, but to produce early feedback about the design of an interactive system before expensive decisions have been made. In the main these techniques are intended for use by human factors experts. They operate on some informal representation of the design, for example a storyboard or a draft of the user manual. The description is rarely sufficient to provide a specification adequate as a basis for programming the system.

Two issues are important in the use of these informal methods. Firstly, because the method is informal it will be more difficult to perform it systematically and therefore ensure coverage. Secondly, the method relies on application by those who have sufficient expertise to apply it correctly, for example interpreting any questions that are asked of the design appropriately.

Although the aim of these methods is that they should be cheap to apply, their cost of application depends on the expertise of those who do the application. They have in common the goal to be able to reason about the usability characteristics of systems during the development stages of its design. Deferring the quality assessment until a prototype of the system is available, can become too costly. The possibility of verifying the specification of the system in the early stages of the design process could reduce the number of changes in the future and thereby reduce the overall cost of development.

Informal methods vary as to whether work representations, that is descriptions of what users are intended to do, are involved in their application. Two classes of methods shall be briefly discussed. The first, *usability inspection*, assumes no representation of the user or the work that is intended to be performed by the device under analysis. Instead it involves systematic interrogation of the design representation using a series of standard questions. *Cognitive walkthrough* on the other hand requires an initial model of how the device is to be used. Again questions are asked systematically of the device but these questions are related to the task representations.

2.1 Usability Inspection

The first class of methods involves systematic inspection of the design by means of guidelines for good practice in the design. It is assumed that there are a number of general characteristics that all usable systems should exhibit. In [12], a usability inspection method (Heuristic Evaluation) is proposed based on this type of approach. Applying heuristic evaluation involves setting up a team of evaluators to analyse the design of the user interface. To avoid bias in the analysis, the team members should not have been involved in the design process. They should be human-factors experts, although studies have shown that teams of users can also provide useful results [3]. Once all evaluators have performed their analysis, results are aggregated. This provides a more comprehensive analysis of the design.

To guide analysis a set of design heuristics is used based on general purpose design guidelines. The set proposed by [12] comprises nine heuristics: *simple and natural dialogue*; *speak the user's language*; *minimise user memory load*; *be consistent*; *provide feedback*; *provide clearly marked exits*; *provide short cuts*; *good error messages*; and *prevent errors*. Using these heuristics, evaluators will inspect the proposed design in order to assess if relevant guidelines are obeyed. Obviously not all heuristics will be appropriate all of the time.

The method prescribes nothing about how analysis is performed in terms of whether or not the system follows a guideline. Typically some type of informal walkthrough approach is used (see Section 2.2). It is also assumed that the team of analysts will envisage situations in which the system shall be used. In effect the analysts imagine possible work situations implicitly, whereas in the method to be discussed next these situations are made explicit.

2.2 Cognitive Walkthroughs

Heuristic evaluation makes no *explicit* documented assumption about how the system is to be used. Hence a heuristic might invite the team of analysts to consider any situation *at all* where feedback is not given. In practice it may well happen that in certain tasks *feedback* is essential while in other tasks users may have enough understanding of what will happen next for feedback to be superfluous.

A class of informal but *structured* techniques make explicit (to differing degrees) the work or tasks that are to be done in using the device. The consideration of what happens in the context of a particular task is therefore the fundamental unit of analysis. This general class of walkthrough

techniques attempts a more work-based usability analysis by *simulating* how users are expected to behave when faced with the system under analysis.

Cognitive walkthroughs [9] is one such technique. Its aim is to analyse how well the interface will guide the user in performing tasks. User tasks must first be identified, and the model of the interface (the device model) must be sufficiently detailed to cover all possible courses of action the user might take.

Analysis of how a user would execute the task is performed by asking three simple questions at each stage of the interaction: *Will the correct action be made sufficiently evident to users?*; *Will users connect the correct action's description with what they are trying to achieve?*; *Will users interpret the system's response to the chosen action correctly?*. To answer these questions, probable courses of action the users will take must be presumed and documented as so-called user models. Problems are identified whenever there is a "no" answer to one of the questions above.

There are variations of this basic idea that assume more or less expertise from the analyst. Methods span human factors and human computer interaction and those developed in one tradition are barely known in the other. The methods that exist differ in two respects: in the way that items or units from the work description are used and in the types and generality of the questions.

For example, cognitive walkthrough asks three simple questions and is designed to explore usability or learnability, the THEA technique [16] provides a larger set of eighteen questions based around Norman's model of cognition, and focusses on human performance with a particular focus on human error. HEIST on the other hand [8] includes timing and other aspects of context in its very large set of questions. TRACEr invites questions in domain terms relating specifically to air traffic control concepts [18].

3 More formal analytic methods

Informal analytic approaches pose problems for engineers of complex interactive systems. Results are largely dependent on the skills of the evaluators: human factors skills rather than software engineering skills. This is a particular problem in the case of heuristic evaluation where the underlying theory is less explicit within the method. The human evaluators will bring into the analysis process a number of assumptions about user behaviour, and about the usage of the device. The validity of the analysis is limited by these assumptions.

Walkthrough approaches are more structured but may become extremely resource intensive as systems increase in complexity and possible activities that relate to the system become more diverse.

Although formal approaches are generally more limited in their application because of the cost of producing initial models they have the potential advantage that they can: demand more clarity from the analyst about the design description as well as the assumptions that form the basis for analysis; provide a precise description that can be used as a basis for systematic mechanical analysis in a way that would not otherwise be possible; and provide an external representation that may make it possible for some of the processes, currently only within the domain of expertise of a human factors expert, to be performed by an engineer.

Formal approaches bring more rigour and automation to usability analysis at the expense of flexibility. They require a specific type of model and a specific type of verification technique which can limit the analysis to be performed.

Proving properties of interactive systems shares with any other approach to analysis the requirement that there should be a model of the key features that are to be under scrutiny in the analysis. Here a model of interactive behaviour is required which highlights the interactive behaviour of the system while discouraging inappropriate bias in the analysis. A device description is to be explored in the context in which it is used. Instead of analysing all possible behaviours it is necessary to decide how to specify those behaviours that correspond to the way that the device shall be used. This itself narrows the context to those features that the designer considers to be important. Once the device and the rules for specifying its context have been explored it is also necessary to decide under what conditions the use of the device is of concern. It may be appropriate to analyse potentially extreme situations or where certain actions might be discrepant with the expectations of the user. Each of these issues shall be explored in more detail below. Four steps as represented in Figure 1 are involved in the process.

Initially, usability requirements, perhaps based on usability guidelines or heuristics, are identified and used to aid the specification of which parts of the device are to be explored.

Once the initial requirements have been expressed, a model of the interactive device and a set of properties are formulated. It is envisaged that at each stage of this process human factors expertise will be involved in assessing the validity of these properties or models. Models capture assumptions about the user's view of the system and properties capture constraints that if broken represent potential failures in the use of the system.

The next stage is to verify whether the properties hold of the device under the constraints imposed by the environment. For example it may be the case that it is necessary to explore a property that reflects the visibility of actions but only over certain paths corresponding to the likely tasks that are to be performed.

Finally the results are explored with the human factors analyst for their likely consequences. If the answer is pos-

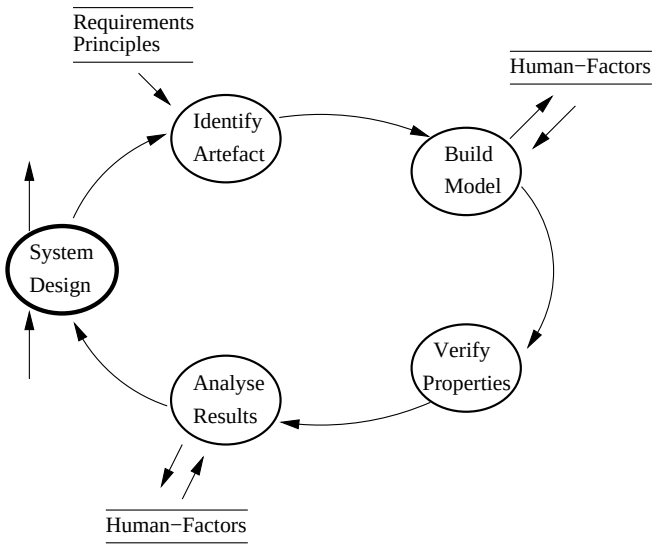

Figure 1. Integration of verification in development

itive then it can be said that under the assumptions used in expressing the model and the property, the system satisfies the usability criteria. If the answer is negative then the reasons why must be investigated. A negative result might indicate that the device model is incomplete. It may therefore be necessary to determine whether all relevant aspects of the device have been considered. Alternatively it might point to a situation where the assumptions made about the user (or the context of usage) must be refined. For example it may be necessary to eliminate specific courses of action from the expected user behaviour. Finally it might be a genuine usability problem.

4 Analysing an interactive device

Formal analysis has been used in a variety of ways to explore interactive systems. In the next sections it will be shown how this type of approach can be useful in reasoning more systematically about usability.

A first approach is to model the device and use knowledge about the user to drive the analysis. This is the approach taken by Campos and Harrison [2]. They use a Modal Action Logic (MAL) based notation for modelling, and a tool that translates the models into SMV (a model checker). Properties can be checked using this system by writing them in Computational Tree Logic (CTL).

Modelling is performed from the point of view of the interaction between device and user and involves taking account of the specific analysis to be performed. To give a

interactor MCP
includes
  aircraft **via** plane
  dial(ClimbRate) **via** crDial
  dial(Velocity) **via** asDial
  dial(Altitude) **via** ALTDial
attributes
  vis pitchMode: PitchModes
  vis ALT: boolean
actions
  vis enterVS, enterIAS, enterAH, toggleALT
  enterAC
axioms
  [] plane.altitude = 0
  [crDial.set(t)] pitchMode′=VERT_SPD ∧ ALT′=ALT
  ...
  [enterAC] pitchMode′=ALT_CAP ∧ ¬ALT′
  per(enterAC) →
        (ALT ∧ |ALTDial.needle - plane.altitude|≤2)
  (ALT ∧ |ALTDial.needle - plane.altitude|≤2) →
                                    obl(enterAC)
  ...

Figure 2. The MCP model

flavour of the notation, an excerpt of a model is presented in Figure 2.

The specification of the interactive behaviour of the device asserts constraints on the way the device is viewed, for example by defining user level actions and the way information about those actions may be perceived (cf. the vis annotations). Further constraints may be imposed by considering the means by which behaviours should take account of contextual factors.

In [2] the mode control panel (MCP) of an aircraft is analysed regarding altitude acquisition. The design of the interface has been based on the plausible assumption that if the altitude capture (ALT) is armed the aircraft will stop at the desired altitude (selected in the altitude dial — ALTDial). This can be expressed as the CTL formula:

AG((plane.altitude < ALTDial.needle ∧ ALT) →
    AF(pitchMode=ALT_HLD ∧
        plane.altitude=ALTDial.needle))

which reads: it always (AG) happens that if the plane is below the altitude set on the MCP and the altitude capture is on then (AF) the altitude will always be reached and the pitch mode be changed to altitude hold.

In the process of checking the properties, counter-examples are generated by the model checker that reflect inadequacies in specification of the model or the property.

In [2] some initial counter-examples are produced that indicate situations where the pilot might take action to stop the aircraft from climbing (for example, changing the vertical speed). When the property is changed to prune those behaviours a new counter-example is produced which indicates that under specific circumstances the aircraft performs an implicit mode change that might lead subsequent user action to cause the aircraft to keep climbing past the target altitude. Palmer [14] reports that a similar problem was detected during simulation.

This counterexample prompts the designer to consider whether there is enough information provided by the device's user interface so that the pilot may be kept in the loop. The designers and human-factors experts can be called upon to clarify the full consequences of the counterexample. How aware will the pilot be of the mode change performed by the automation? Is this issue adequately covered in the manuals, and during training? Should the system be redesigned and how? What engineering constraints come into play regarding the design? Being able to raise these issues against a formal proof background in early design stages will undoubtedly allow for a better/safer design from the start. It will also reduce downstream costs of failing to discover these problems until too late.

5 Modelling the user to limit the analysis

As mentioned the approach above binds constraints about which paths can occur into the properties. There are other ways of restricting only to paths that are relevant. Rushby [17] models simple assumptions about the user's view of the system. The approach can be characterised as embedding elements of a user model into the device model and thereby constraining the behaviours of the system that must be analysed. The specification is modified to capture the changes in the user model as a result of actions by the system.

The problem to be explored using the model checker is whether there are discrepancies between the expectation that the pilot has of the ALT mode and its actual state. Considering the state of the altitude capture as defined by the device mode logic (ALT), and as idealised by the user (idealALT), the property could be expressed as: AG(ALT=idealALT)

A number of questions arise about how the user model is constructed. It must be decided what the user will or not be aware of, and whether the user will remember relevant information. These issues must be dealt with in the context of a model of cognitive behaviour. When the approach is used by software engineers there is a risk that wrong assumptions about cognition may be built into the model through ignorance of the science behind the model. These assumptions, however, are central to the reasoning process and because they are embedded in the specification their analysis becomes less open to discussion and dispute. In the example above, assuming the user does not notice the relevant indicator, then the mode problem is detected when we try to verify the property. When the user is assumed to notice the indicator then the problem does not show up in the analysis.

More elaborate attempts have been made to model what the operator is likely to do with a given device design. Two examples are syndetic modelling [4] which involves modelling the user as a process based on assumptions from a cognitive architecture (Interactive Cognitive Subsystems) and programmable user modelling (PUM) [19] which uses a planning model. In both cases, the analysis is based on describing the joint behaviour of device and user models. While the primary goal of the syndetic modellers has become to use formalism to make assumptions made in cognitive models more explicit and thereby more precise, PUM assumes that a cognitive architecture is programmed with domain and device specific knowledge. The aim of PUM therefore is to better understand and design the interactive system.

In this case, instead of enriching the device model with a (state-based) model of user's knowledge, a separate model of the knowledge needed by the user to operate the device is first developed, and then combined with the device model in order to investigate the behaviour of the resulting system. An *instruction language* is used to describe the goals of the user in using the device as well as the operators or methods as understood by the user to achieve these goals. Unlike the approach above these operators do not necessarily correspond to device level operations. Conceptual operators are defined in terms of pre- and post- conditions on the state as idealised by the user.

Once the models are fully developed a planning engine is used to show how the assumptions made in the model lead to a sequence of actions at the device level that meet the goals expressed at the domain level. This is typically done using means-ends analysis. The execution of a PUM is therefore an attempt to mimic how a rational user will behave when faced with the device. In most of the PUMA work as represented by [1] this process is ultimately performed relatively informally by hand.

6 Using the task to drive the analysis

In previous sections two approaches have been discussed for analysing an interactive system. In the first approach the model of the device is the primary focus for exploration. Assumption about context and user may be embedded within properties implicitly. In the second approach one model is used to describe the device in terms of the methods or actions that are available to users as understood by users, and a second model describes the goals as understood by

users and encodes assumptions about cognition. Task analysis is a third approach. The motivation is similar to the other two in that the aim is to generate sequences that can be used as a basis for analysing the device. Task analysis however is a discipline that involves observing work as practiced, interviewing users, developing training material in order to understand the goals, sub-goals and actions as perceived by the users. The task analyst then represents the tasks that have to be carried out in terms of the goals, sub-goals and procedures or plans that should be followed. The problem with this approach is the use of the word "should". The approach is normative and therefore may ignore important classes of activity that were unforeseen.

A variety of representations of tasks have been explored. Either using specific languages such as ConcurTaskTrees (CTT) or using more general languages, for example Ofan and *Mur*ϕ. These task representations are used in a variety of ways to analyse the interface. Paternò in [15], develops an interface model using LOTOS which is derived from a task description. The model therefore reflects the structure of the tasks that the device is supposed to support. More recently Paternò has used CTT to model and analyse the tasks models directly, no device model is used. Normative behaviour drives the analysis completely and properties such as *reachability* are analysed only in terms of the paths permitted by the task description.

Fields [5] has produced a more encompassing approach in which separate device and task models are developed, and the behaviour of their combination is analysed. In this case the analysis, while based on the normative behaviour represented by the task, systematically generates sequences that are based on perturbations of these normative task paths using simple mutations: omission, commission, repetition, reversal and so on, as used in hazard analysis techniques. Device models are written in the *Mur*ϕ language, and task models in a special purpose notation which is automatically translatable into the *Mur*ϕ language.

A further approach uses Ofan Statecharts which encode not just the device but also the task and the environment. These state charts are rendered checkable using the SMV model checker [10].

It would be straightforward to add a task model to figure 2. Whatever language is used, task models will describe the intended traces of behaviour. If only user actions are considered the actions in the task model will be the subset of visible actions of the device model, and the task model describes the intended traces of user behaviour. A possibility to representing those traces is using C/E Petri nets [13]. For example, the Petri Net in figure 3 represents the task of setting an altitude and toggling the altitude capture.

The Petri Net can be translated into an interactor for analysis. Each place is modelled by a boolean state variable representing whether it is marked. Each transition is modelled

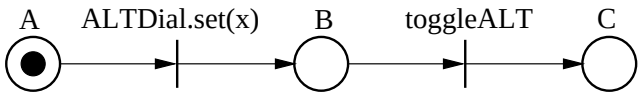

Figure 3. A simple Petri Net

by two axioms. A permission axiom stating when the transition is allowed to fire, and a modal axiom stating the effect of the transition on the marking of the net. For example, transition toggleALT is modelled by axioms:

per(toggleALT) $\rightarrow$ (placeB $\wedge\neg$placeC)

stating that it can fire when place B is marked and place C is unmarked (i.e. after ALTDial.set()); and

[toggleALT] $\neg$placeB' $\wedge$ placeC' $\wedge$ placeA'=placeA

stating that after toggleALT being fired place B becomes unmarked, place C becomes marked, and the marking of place A does not change.

The task model represented by the Petri Net can be combined with the device model binding task and device visible actions together. Properties can be checked of the form AF(goal state of the system) (the goal can be achieved by performing the task) or AG($\neg$hazardous state) (no hazardous state can be reached when performing the task).

7 Discussion

Ensuring the usability of a system is a complex and difficult endeavour. It has been shown elsewhere [3] that it is possible to reason about designs, from models of those designs, in order to improve usability prior to actual implementation and deployment of the system. Typical approaches, however, lack the rigour and thoroughness that only formal (mathematical) models can provide.

When attempting to model an interactive system for analysis we are faced with the problem of how to capture the diversity of concepts and concerns that are involved in the analysis of usability. Three basic aspects can be identified. The designed device, the user(s), and the work to be carried out (usually expressed in terms of tasks or goals). In a good interactive system design the device will support the user in carrying out the work. The quality of such support can be measured in many ways: efficiency, effectiveness, enjoyability, etc.

We have presented a number of proposals for the systematic analysis of usability that use different combinations of these ingredients. In the approach of [2] only the device is explicitly modelled. Considerations about user and work are used to drive the analysis. This style of unconstrained approach is best suited to detect problems with unpredicted

device, or user, behaviour. This can be especially relevant for complex systems where tasks can interact in unexpected ways. It does not, however, provide much direct support when investigating how users will actually use the system. That step is left outside the formal reasoning phase. Additionally, some of the behaviours identified by the automated analysis might not be relevant to the real system. This analysis is also to be done outside the tool.

To address these issues, authors have proposed the use of user models in conjunction with the device models. This directly introduces human factors issues into the models, thus lessening the dependency of the analysis on human-factors expertise, and making it more accessible to software engineers. The general idea behind this style of approach is to set a specific goal, and see whether the *system* can find a way to satisfy it. This is specially interesting when investigating how a user will interact with a device in order to achieve a certain goal. In this aspect, it can be related back to Cognitive Walkthroughs. However, the planner may generate unexpected solutions that might lead the analyst to appreciate unforeseen consequences and scenarios that would not be otherwise explored.

Adding the user model means adding assumptions about user behaviour which will restrict the possible behaviours of the device. This bias in the analysis can lead to key issues in the device's design being ignored. User model architectures such as PUM help in guiding the process and in spelling out what assumptions are being made.

An alternative is to model, not the user, but the tasks that the user must perform. The approach will not give much direct support when investigating how easy users will find performing the tasks to be, but enables the analysis of how the device supports the proposed tasks. This can be done both in terms of whether performing the task achieves the desired goal, and of whether performing the task might lead the device into unwanted states. The consequences of erroneous behaviour are investigated by introducing errors in the task model.

It has become clear that the different styles of approach are complementary in that they support different styles of reasoning about the system. Unconstrained device analysis enables a more thorough exploration of the device. The interpretation of the results must then be performed with the aid of human-factors experts. By adding user models to the analysis one aims at analysing how users will be able to use the device. Using task models one analysis how the device supports the tasks the user is expected to perform.

8 Conclusions

A number of studies indicates that human factors account for a very large proportion of failures in systems [7, 11]. In the past, software development methods have attached relatively little significance to human factors issues. Unless that is changed the proportion of failures attributed to human factors will keep increasing as other aspects of software development improve. Clearly there is the need for a better integration of human factors concerns into the software engineering life cycle.

One specific facet of such integration is the need to reason about the usability of systems' designs from early in the development process. Usability analysis must not be left to the latter stages of software development when changes will be more difficult and expensive to make. Instead, a number of lightweight methods are needed that enable reasoning about the usability of systems from the early stages of design, thus enabling the design to be shaped by the usability criteria and concerns.

A number of discount (analytic) methods for the analysis of usability have been proposed, and studies have shown that they can be useful in detecting potential usability problems [3]. Discount techniques have distinct advantages over empirical methods. They are cheap to use, they do not require extended advanced planning and they can be used in the early stages before a design is implemented.

There are however problems when these techniques form part of a process where it is intended that they be applied by software engineers because they require human factors expertise. Another problem is the sheer size and complexity of the models that must be considered when it comes to complex systems. Usually the methods tend to focus on what could be called surface issues in the interaction, with little consideration of more complex behavioural issues of the interaction between user and device that will arise in real usage conditions. In this respect it can be argued that not even empirical evaluation can guarantee absence of such errors since for complex systems it is usually not viable to test all possible usage conditions/situations.

This paper presents some recent proposals of more formal and systematic usability analysis methods that attempt to provide answers to these problems. The claim is that these techniques, although narrower in scope, provide a more thorough analysis in which human factors claims are more clearly identified and substantiated using complementary expertise from other parties to the design process. Hence software engineers carry out part of the process but there are well defined stages where human factors input is required. In an initial phase human factors expertise is brought in to help select and model relevant system features, the analysis can then progress in a more typical software engineering setting, finally the analysis of the results must go back to human factors expertise.

The problems with using formal techniques relate to how well the techniques scale; how easy is it to use the methods; and whether the methods bias the analysis so that key issues may be ignored. We do not contend that this type of

approach is the answer to all of the usability engineering problems. We do however feel that it has a role to play during interactive systems design and analysis. In the context of design problems where there is no obvious pay off using formal techniques, for example where more exploratory techniques are currently used, it might be argued that costs of applying these techniques are too high. However the situations where early analysis can reduce substantial downstream costs, either because the system is safety critical or because the cost of shipping volumes of less than usable systems cannot be countenanced, are increasing. More in depth and realistic studies are required to justify these claims.

References

[1] Ann Blandford, Richard Butterworth, and Jason Good. Users as rational interacting agents: formalising assumptions about cognition and interaction. In M. D. Harrison and J. C. Torres, editors, *Design, Specification and Verification of Interactive Systems '97*, Springer Computer Science, pages 45–60. Springer-Verlag/Wien, June 1997.

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

[3] Heather W. Desurvire, Jim M. Kondziela, and Michael E. Atwood. What is gained and lost when using evaluation methods other than empirical testing. In A. Monk, D. Diaper, and M. D. Harrison, editors, *People and Computers VII*, pages 89–102. Cambridge University Press, September 1992.

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

[5] Robert E. Fields. *Analysis of erroneous actions in the design of critical systems*. DPhil thesis, Department of Computer Science, University of York, 2001.

[6] M. Harrison and H. Thimbleby, editors. *Formal Methods in Human-Computer Interaction*. Cambridge Series on Human-Computer Interaction. Cambridge University Press, 1990.

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

[8] B. Kirwan. *A Guide to Practical Human Reliability Assessment*. Taylor and Francis, 1994.

[9] Clayton Lewis, Peter Polson, Cathleen Wharton, and John Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *CHI '90 Proceedings*, pages 235–242, New York, April 1990. ACM Press.

[10] K. Loer and M. Harrison. Formal interactive systems analysis and usability inspection methods: Two incompatible worlds? In P. Palanque and F. Paternó, editors, *7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS)*, volume 1946 of *Lecture Notes in Computer Science*, pages 169–190. Springer-Verlag, 2001.

[11] D. MacKenzie. Computer related accidental death: an empirical exploration. *Science and Public Policy*, 21(4):233–248, 1994.

[12] J Nielsen. *Usability Engineering*. Academic Press, Inc, 1993.

[13] Ph. Palanque, R. Bastide, and V. Senges. Task model - system model: towards an unifying formalism. In *Proceedings of HCI International conference*, pages 489–494, Yokoohama, Japan, July 1995. Elsevier.

[14] Everett Palmer. "Oops, it didn't arm." - a case study of two automation surprises. In Richard S. Jensen and Lori A. Rakovan, editors, *Proceedings of the Eighth International Symposium on Aviation Psychology*, pages 227–232, Columbus, Ohio, April 1995. Ohio State University.

[15] Fabio D. Paternò. *A Method for Formal Specification and Verification of Interactive Systems*. PhD thesis, Department of Computer Science, University of York, 1996.

[16] S. Pocock, M. Harrison, P. Wright, and P. Johnson. THEA: A technique for human error assessment early in design. In M. Hirose, editor, *Human-Computer Interaction INTERACT'01 IFIP TC.13 International Conference on human computer interaction*, pages 247–254. IOS Press, 2001.

[17] John Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, February 2002.

[18] S. Shorrock and B. Kirwan. Development and application of a human error identification tool for air traffic control. *Applied Ergonomics*, 33:319–336, 2002.

[19] Richard M. Young, T. R. G. Green, and Tony Simon. Programmable user models for predictive evaluation of interface designs. In K. Bice and C. Lewis, editors, *CHI'89 Proceedings*, pages 15–19. ACM Press, NY, May 1989.