Knowledge Media Institute

Training Software Engineers in a Novel Usability Evaluation Technique

Ann Blandford, Simon Buckingham Shum and Richard Young

KMI-TR-23

May, 1996 • Updated July, 1998

International Journal of Human-Computer Studies (1998, in press)

Training Software Engineers in a Novel Usability Evaluation Technique

Ann E. Blandford
School of Computing Science, Middlesex University, Bounds Green Road, London, N11 2NQ, U.K. Email: A.Blandford@mdx.ac.uk

Simon J. Buckingham Shum
Knowledge Media Institute, The Open University, Milton Keynes, MK7 6AA, U.K.
Email: S.Buckingham.Shum@open.ac.uk

Richard M. Young
Department of Psychology, University of Hertfordshire, Hatfield, Herts. AL10 9AB, U.K.
Email: R.M.Young@herts.ac.uk

Abstract

Novel approaches to designing or analysing systems only become useful when they are usable by practitioners in the field, and not just by their originators. Design techniques often fail to make the transition from research to practice because insufficient attention is paid to understanding and communicating the skills required to use them. This paper reports on work to train software engineering students to use a user-centred language for describing and analysing interface designs called the "Programmable User Model Instruction Language", or IL. Various types of data, including video, students' IL descriptions and brief usability reports were collected during training, and subsequently analysed. These show that after 6 hours' training, students have a good grasp of the syntax of the notation, and are starting to use notational affordances to support their reasoning, but that their reasoning is still limited by a poor grasp of the underlying cognitive theory. A comparison of the analyses of trainees with those of experts provides a means of developing a better understanding of the nature of expertise in this area — as comprising an understanding of the syntax and the surface semantics of the notation, the underlying cognitive theory, the method of conducting an analysis and the implications of the analysis for design.

1. Introduction

When a novel HCI technique is developed, one of the important issues to be addressed is how the technique can be transferred from the research to the practitioner community. Inevitably, as the work on Cognitive Walkthroughs has shown over the past few years (Lewis, Polson, Wharton & Rieman, 1990; Polson and Lewis, 1990; Wharton, Rieman, Lewis & Polson, 1994), this is unlikely to be a single event in which a technique, having been developed to maturity, is then handed over to the practitioners to be instantly taken up and exploited. There is an iterative cycle of activity in which developers learn to take a practitioner-centred view of their technique, and adapt it in the light of experience, while practitioners learn about the technique and its scope and how to apply it.

The main aim of the work reported here is relatively modest: to investigate how trainee software engineers use the "PUM Instruction Language" (IL) after a day's training. Studying trainees is necessary to develop an understanding of what actually happens when one learns to use a notation such as IL. Video data of trainees working with IL allows us to address issues such as what concepts beginners struggle with and how IL constructs can contribute to design insight. As John and Packer (1995) note, you can only distinguish with any certainty between the power of a technique and the contribution of the analyst's craft skills when you have non-originators using a technique. Process data can then show what analysts did and how they

gained insights into the design problem. One of the aims of this study is to address these points. A further issue is whether using IL makes a discernible difference to the kinds of usability assessments made. This is important, particularly in relation to understanding how much of the leverage of a particular technique derives from the technique itself, and how much from the skill of the analysts using it. The study also raises generic issues about the transfer of theoretically-based HCI evaluation techniques to practitioners.

To investigate how trainees learn to use IL, we developed a study that involved training a group of subjects to write IL descriptions and collecting a range of data from them. The tutorial material was developed to be suitable for delivery in one day (6 hours). This material includes both presentation material and student exercises based on example systems. This time of 6 hours was chosen to be comparable to that which one might reasonably be expected to spend if delivering a tutorial at a conference, or teaching about user modelling as part of a university course that includes a reasonable HCI component. The training was given to a group of 16 students taking a course in Human-Computer Interaction (HCI) at the Free University of Amsterdam. As well as the 6 hours allowed for training, additional time was allocated to the collection of experimental data, as described below (Section 2).

1.1 A FRAMEWORK FOR UNDERSTANDING THE ROLE OF MODELLING IN ASSESSING DESIGNS

Observation (by Buckingham Shum) of expert HCI modellers, combined with our own reflections on using particular approaches, has led us to propose a framework for thinking about the process of modelling in relation to design and evaluation, as shown in Figure 1. This framework incorporates three key links which enable a modeller to move iteratively from a problem, to a modelling expression of that problem, to design insights and recommendations. It allows us to break down the process of evaluation and re-design into identifiable stages that can be considered separately.

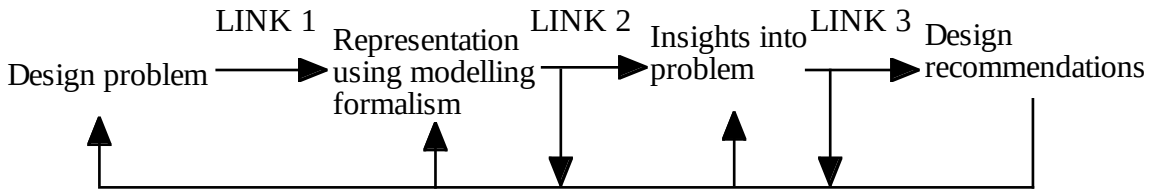

Figure 1: Key links to negotiate in a model-based analysis

The process of describing a design problem using a particular representation (link 1) helps an analyst to acquire a better understanding of the problem, and hence gain insights into it (link 2), which in turn can help generate recommendations for design changes (link 3). This framework can serve two key functions: firstly, to help establish to what extent a modelling approach is contributing substantively to the reasoning process, and secondly, to make the approach transferable to other analysts by encoding the process into a more systematic, reusable form.

Link 1 is the primary focus of the study reported here: how do trainees acquire expertise with the IL formalism? *Link 2* reflects expert modellers' ability to recognise the *significance* of different notational patterns or states in a model; that is, make the mapping from the formalism back to the real world. Link 2 related training questions include: to what extent do the trainee modellers negotiate Link 2, and are there any differences in the nature of the analyses reported that can be attributed to IL as opposed to 'craft skill'? Finally, *Link 3*, moving from insights to design recommendations, is where creative design is needed; recommendations rarely flow directly from modelling, as other factors that do not fall within the scope of the formalism often come into play at this point. Drawing on modelling case studies with designers, we have discussed how this link can be negotiated as modellers seek to communicate model-based insights to designers who are non-modellers (Bellotti *et al.*, 1995; Buckingham Shum *et al.*, 1996). As we explain below, it is unlikely that IL-beginners will gain great insight into a problem due to IL, when they are at the same time learning it, so this link was of less

importance at this stage. We revisit Figure 1 in our conclusions, in the light of the study's results.

1.2 ISSUES IN TRANSFERRING THEORY-BASED HCI DESIGN TECHNIQUES

There are various concerns that are pertinent to transferring any novel, theory-based design technique from the research laboratory to practitioners. One is that of understanding the roles of the formal notation and the underlying theory in supporting reasoning. Another is establishing effective ways of encapsulating and communicating the skill of modelling.

The roles of the notation and the underlying theory

A design or analysis technique never stands as simply a notation, method or tool. It derives from, and encourages its user to adopt, a particular orientation to design by virtue of the domain constructs that it addresses (such as people, work, artefacts), the terms in which it does so (e.g. as information processors; as social beings; as formal or unformalisable agents), and the kind of analysis which it facilitates. In this respect, a notation can help to support or focus an analysis. One key question which needs to be addressed is how much general disciplinary and method-specific training one actually needs in order to gain some benefit from using a technique.

Cognitive Walkthrough (CW; Wharton *et al.*, 1994) is one technique where the issue of divorcing theory from practice has been important. CW is not a modelling technique, but a structured methodology for critiquing a user interface. It seeks to deliver the power of a cognitive model of learning (CE+; Polson & Lewis, 1990) in the form of a checklist of principles without actually requiring analysts to know about the underlying theory. Several studies of CW have now been reported, which together are an excellent example of the use of empirical data to drive the design of a technique to meet the skills of its target users and the constraints of realistic design practice (Lewis, et al., 1990; Rieman, Davies, Hair, Esemplare, Polson & Lewis, 1991; Polson, Lewis, Rieman & Wharton, 1992; Rowley & Rhoades, 1992). The CW technique has been extensively revised in the light of evaluations over several years. This work illustrates well the nature of the problems faced in developing a theoretically based technique targeted at practitioners not trained in the underlying theory. Having set out to develop a cognitive science-based technique for non-cognitive scientists, they encountered unforeseen problems such as misconceptions about the meaning of 'tasks' and 'goals', and analysis at an inappropriate level of detail. Training in the Walkthrough methodology now includes more extensive introduction to the underlying theory than was originally thought necessary.

One aspect of the investigation into the learnability of the IL is how dependent analysis is on a deep understanding of the underlying theory (notably the means-ends planning described in Section 1.3). Another aspect is understanding what role the notation itself plays in supporting usability reasoning. Since the IL is a "semi-formal" notation that can be used with varying degrees of rigour (as described below), an additional issue is how to achieve an appropriate balance between constraining the analysis unnecessarily, and allowing so much freedom that using the notation no longer requires the discipline of rigorous analysis which can yield insight in tackling difficult usability issues. Within HCI and related research, there is a growing recognition of the cognitive and social implications of requiring users to structure their expression of ideas in formal ways (e.g. Pedersen & Suchman, 1991; Shum, 1991; Shipman & Marshall, 1994). Formality permits machine interpretation and manipulation of information at a finer grain of detail, but comes at the cost of a wider translation gulf for humans, for whom formal notations are an unnatural form of expression compared to natural language. This persistent cost-benefit trade-off is as relevant to IL as to any other formalism. With respect to the issue of where on the formality continuum one should locate IL-usage, we consider two perspectives: the role of rigorous formality for trainee designers, and the practical demands of a formalism's use for communication during design.

Identifying and encapsulating craft skill

Another issue to be addressed is how developers of a new technique can identify the implicit skills and knowledge that they have gradually developed, in order to help new users negotiate the initial block which so often characterises, and sometimes sabotages, early attempts to use a new formalism.

One way to encapsulate such craft skill is through informal heuristics which are simply collections of the kinds of tips which an expert might give to a beginner. For example, training materials for other design notations such as *QOC* (MacLean, Young, Bellotti & Moran, 1991) include heuristics that encapsulate hands-on craft skills. Similarly, Kieras (1988) describes work to make Cognitive Complexity Theory (Kieras & Polson, 1985), as based on GOMS modelling (Card, Moran & Newell, 1983; John & Kieras, 1996), a practical user interface design technique. Kieras recognises that CCT and GOMS, despite their formality, leave decisions to the analyst which can make a significant difference to the validity or usefulness of the analysis. He therefore provides some guidance on such issues as making 'judgement calls' on user cognition in the absence of reliable data, deciding what tasks to analyse, when to conduct a GOMS task analysis, and GOMS-based criteria for evaluating and improving the design.

Drawing on such experiences, the IL tutorial included a number of heuristics. Students were supplied with a handout listing these heuristics, and they were also alluded to regularly during training. In addition, the training included a gradual introduction to the basic concepts and techniques needed for writing and analysing an IL description, and understanding was developed through the use of worked examples, as illustrated below.

1.3 INTRODUCTION TO PUM IL

The Instruction Language (IL) was developed as a programming language for a Programmable User Model (PUM). The PUM is a cognitive architecture which, when programmed with knowledge about a particular device, can yield predictions about likely user behaviour with that device. Therefore the IL is a language for describing the knowledge about a device that a prospective user needs. The details of the PUM are reported elsewhere (Young *et al*, 1989; Blandford and Young, 1995). The IL is presented below. The process of conducting a full IL analysis can be broken down into 5 stages, as follows:

1. The first stage is to identify candidate task scenarios; these are ideally frequently performed tasks, or tasks that the analyst informally anticipates might be problematic.
2. The second stage is to identify conceptual objects and operations that the user will need to work with in performing the tasks.
3. The third stage is to express this knowledge of objects and operations in terms of the Instruction Language. At this stage, a corresponding model of the device should also be constructed. Difficulties are frequently found at this stage: it may be very hard to express the user's knowledge or inconsistencies between the user's knowledge and the device representation may be found. If great difficulties are found, analysis may finish here, with a report back on inconsistencies or on knowledge the user needs that is not readily available.
4. If a satisfactory consistent description has been produced then the analyst proceeds to hand-simulate the modelled behaviour of the interactive system; this may help identify further sources of difficulty.
5. Finally, if required, and if the model is sufficiently specified, an automatic simulation can be run to identify likely user behaviours with the system.

When used as a programming language to generate a running model, as in Blandford and Young (1993, 1995), the IL has to be written rigorously and formally. When used as a tool to aid thinking, for a less full and formal analysis in which the *process* of constructing and reasoning with the IL description is more important than the product, the IL description can be less formal. In the study reported here, we focused attention on stages 2-4; for all examples, stage 1 (problem definition / task statement) had already been done as part of the preparation

work, and we did not aim to develop running models with the students. Therefore, we trained the students to use a semi-formal version of the IL.

Here, we give an introduction to the IL, and an example of its use, to support the discussion of method and results that follows.

Summary of Instruction Language

The IL is a declarative language for describing the knowledge users need to achieve any task with a device. The IL analyst's job is to describe what the user needs to know, and how any necessary information is made available to the user, so that the user can both perform the task and know when the goal has been achieved.

The IL description consists of three parts:

- a set of declarations, listing

— the *conceptual* objects that the user is manipulating when working with the device (listing all object types, and maybe enumerating named objects), and

— *relationships between those objects*, in the form of functional relations between objects of named types and also predicates that may hold true for objects of named types. For all relations, the analyst should declare whether or not the user can find out values of the relation by looking at the display.

- a description of the user's knowledge, consisting of

— *conceptual operations* (see below),

— *the user's initial knowledge*, in terms of functions and predicates that are declared to hold true (note: this is the user's knowledge, not necessarily the device state), and

— the user's *task*, in terms of functions and predicates that should hold true in the goal state.

- a device description, listing

— the *device commands* that are available, in terms of the way the device state changes as a result of the user issuing each command,

— the *initial device state*, in terms of functions and predicates that hold true in that state, and

— what information is displayed to the user.

One aspect of this description — that of conceptual operations — merits further expansion. Each operation is defined in terms of:

- the types of the objects involved in the operation (the *arguments* of the operation).
- the reason that a user would select this operation, in terms of functions and predicates the operation is relevant for achieving. This is the *user-purpose* of the operation.
- conditions that must hold true before the operation can achieve its user-purpose, and that the user would try to make true so that this operation can be invoked. These are *preconditions* for the operation. As discussed below, preconditions are conditions that the user might adopt as sub-goals, so for every precondition there must be another operation which has that condition as its user-purpose.
- conditions, called *filters*, that must hold true before this operation is considered for addressing its user-purpose. This includes conditions that the user cannot change (those for which there is no operation having that condition as its user-purpose), and conditions that the user would not make true simply to invoke this operation. For example, in the Mailtool example discussed in more detail below, one of the operations is to highlight a particular mail message in the current folder. For the tasks being considered, it would not make sense for the user to move a message to this folder just in order to highlight it, so the condition that the message is in the current folder is a filter, not a precondition.
- any effects of operations that users are expected to mentally track (i.e. predict, or anticipate, without necessarily checking that the effect has been achieved when the

corresponding device command is issued). As a general heuristic, *tracked effects* are those that are predictable to the user and that match the user-purpose of the operation. (So side-effects are not generally tracked, and neither are the effects of operations for which the user does not have enough information to predict the effect on the device.)

- the corresponding *device command*.

One of the important considerations for the analyst constructing an IL description is how users know all the things they need to know in order to achieve their goals with this device. There are three sources of information: facts that the user knew at the beginning of the interaction and that do not change (e.g. the user of an automated teller machine must know the relevant personal identification number), information that is available from the display and information that must be tracked to be known. In many cases, an analysis that focuses just on this concern will highlight many of the usability problems with a device. In some instances, additional insights might be gained from hand-simulating the problem solving and external behaviour of the modelled user. Such hand-simulation can also be a good way to validate the IL description.

The simplest modelled behaviour is means-ends problem solving that takes account of changes in the device state. The user identifies operations to perform on the basis of "selection by purpose" — by identifying operations whose user purpose matches the current goal and whose filters are satisfied. If any preconditions are not satisfied, they are adopted as subgoals. When an operation with satisfied preconditions is selected, the user issues the corresponding device command and the user's knowledge of the device state changes in response to tracked information and perceived changes to the device state. This simplified account of the modelled user's cognition is sufficient to validate the IL descriptions produced by subjects in the study. It is too simple for modelling more complex behaviours.

Example of using IL

To illustrate the use of the IL, we take a brief example from the training material presented to students. This example was designed to introduce various IL concepts and their use. It concerns a chocolate vending machine. This machine offers a choice of three different kinds of chocolate bar. First the user puts in the money (50p), then she presses one of three buttons to choose the kind of bar she wants, then she takes the bar.

To encode this situation, we need to consider what the hungry user needs to know in order to satisfy her goal of eating a bar — say a Venus bar.

The IL description is built around the *conceptual objects and operations*, i.e. the things and actions that the user knows about. The relevant things are chocolate bars, coins, and so on. We classify each object as being of a particular *type*, and then we can start writing:

```
OBJECTS
  coin: 10p, 20p, 50p, …
  chocolate-bar: venus-bar, mars-bar, pluto-bar
```

To describe where the objects are, what they are like, how they relate to each other, and so on, we write *predicates* and *functions* that describe what the user knows about a situation, i.e. about named relationships holding between particular objects. In this particular case, the most important relationships are all expressed as predicates. For each, we also state *how* the user knows whether or not a predicate holds:

```
PREDICATES
  user-has(chocolate-bar)          -- detected by looking
  in-machine(chocolate-bar)        -- (assumed always true)
  user-has-cash(coin)              -- detected by looking
  money-paid(coin)                 -- (must be tracked)
  inside-user(chocolate-bar)       -- detected by feeling!
```

We can now talk about the user's knowledge of the initial state:

```
INITIAL-STATE
      user-has-cash(50p)
```

There is room for some informality in this description. For example, the assertion `user-has-cash(50p)` might mean that the user knows she has (exactly) 50p of cash, that she has *at least* 50p of cash, or that she has a 50p coin. As long as the user has whatever coins the chocolate machine needs, the meaning can be left imprecise.

Just as we can use predicates to describe the initial situation, we can also use them to describe the desired situation. We encode the user's goal of wanting to have eaten a Venus bar as:

```
TASK
  inside-user(venus-bar)
```

We turn now to the conceptual operations that describe what the user can do. In this case, we consider three operations: paying for, selecting and then eating a bar:

```
OPERATIONS
pay-for-bar()
      user-purpose:     money-paid(50p)
      filter:           user-has-cash(50p)
      tracked:          money-paid(50p)
      device-command:   insert-money(50p)

select-bar(chocolate-bar: C)
      user-purpose:     user-has(C)
      precond:          money-paid(50p)
      device-command:   press-button(C)

eat-bar(chocolate-bar: C)
      user-purpose:     inside-user(C)
      precond:          user-has(C)
      device-command:   chomp-chomp!
```

For each operation, we have specified what its purpose is and what the corresponding device command is. Since the `eat-bar` operation does not involve interaction between the user and the machine, we give it a jokey "command". For two of the operations (`select-bar` and `eat-bar`), we have specified *preconditions*, which are conditions that must hold before the operation can achieve its purpose. For the third (`pay-for-bar`), we have specified a *filter*; this is also a condition which must hold before the operation can achieve its purpose, but is one that cannot be achieved within the closed world of this analysis, because we do not specify how she might go about obtaining any cash.

We have also specified that the `money-paid` is *tracked* by the user. This means that the predicate is remembered by the user. After this operation has been done, the user then *knows* that the money has been paid. This is important, because the vending machine does not visibly change state in any way — for example, it does not have a counter displaying how much money it has received — so if the user did not track the information, she would not know that the money had been put into the machine, and perhaps might put 50p in again, … and again ...

We have described the device informally; we need to make that description slightly more rigorous:

```
INITIAL DEVICE STATE
  in-machine(venus-bar)
```

```
  in-machine(mars-bar)
  in-machine(pluto-bar)

DEVICE COMMAND
insert-money
      has effect that machine is ready to dispense one bar. Device state
      does not change visibly.
press-button(C)
      has effect that the machine dispenses bar of type C. Bar is visible.
```

We are now in a position to trace out by hand how the specified model will behave. The model has two core principles that drive its behaviour:

Selection by purpose: If the user has a goal G, and if there is an operation O with a user-purpose that matches G and whose filters are satisfied, then the model will choose O — or one of those operations, if there are several.

Precondition subgoaling: If the model has chosen an operation O, and if it has precondition(s) P that are not satisfied, then the model stays committed to executing O but first sets up the goal(s) of achieving P.

Figure 2 illustrates how the modelled user acquires goals and commitments, and selects actions to achieve goals. Starting at the top-left and moving down, the figure shows how goals and commitments get adopted by matching preconditions and user-purposes. Once the user becomes committed to an operation (pay-for-bar) whose preconditions are satisfied, the user can start acting (shown in ovals), and through tracking and visible effects the user's knowledge of the state of the device is updated until (top right) the goal of the task is achieved.

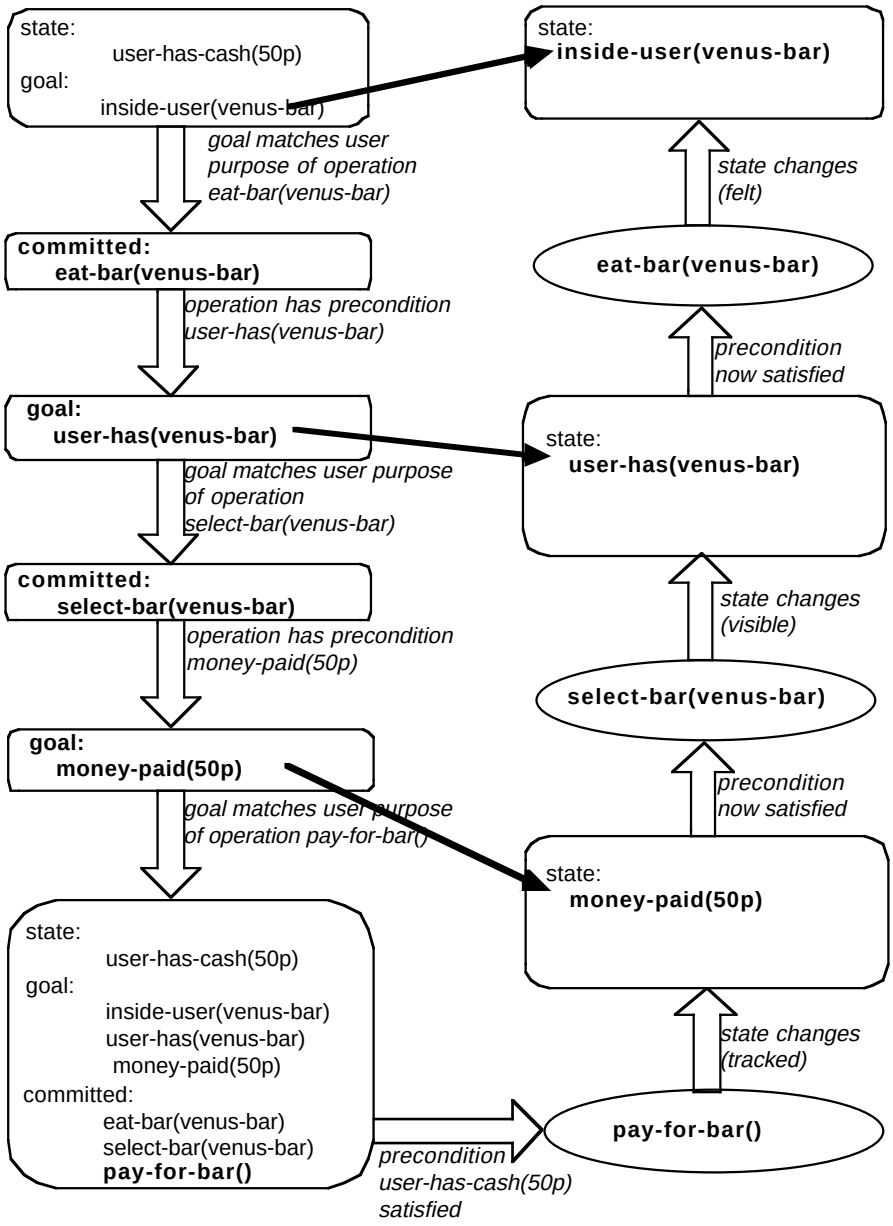

Figure 2: modelled behaviour for the vending machine example.

Constructing an IL description

As illustrated in the example above, familiarity with the syntax alone is not enough to enable novices to write useful IL descriptions. For example, they need to understand how to reason with preconditions and filters, and to trace what the modelled user knows. The ability to assess their own descriptions critically is also needed. The page of heuristics given to students summarised the important points regarding the process of writing a description (e.g. that it is an iterative activity) and the syntax (e.g. that one way users gain knowledge of the state of the device is by *tracking* the effects of operations; that the user model might select an operation whose *preconditions* were not satisfied, and would sub-goal to achieve a precondition, but would not select that operation if a *filter* was unsatisfied). These heuristics focused on the process of constructing an IL description, rather than hand-simulation or reasoning with the description to analyse usability difficulties.

2. Method

The study was based around 6 hours' training, delivered to a group of 16 HCI students. Most of them (13 out of 16) were trainee software engineers; the other 3 were psychology students. Participants received free lunches and were given course credit for attending. There was no drop-out; all subjects who attended the training contributed also to the data collected.

Subjects were asked to work in (self-selected) pairs for much of the time, and these pairs were each allocated, at random, to one of two groups. These groups worked on different test examples. Before the training, subjects discussed one of the test examples in pairs, then individually wrote brief usability assessments of that example. Following the training, subjects wrote IL descriptions (in pairs) and brief usability assessments (individually) of the other example. Finally, students filled in feedback forms indicating their views on the IL and the training. This design is summarised in table 1.

Approximate duration	Pairs 1 - 4	Pairs 5 - 8	Data collected
20 minutes	pre-training analyses of "Windows" example	pre-training analyses of "Mailtool" example	pre-training reports
6 hours	training on IL	training on IL	
1 hour	write IL description of "Mailtool" example	write IL description of "Windows" example	videos of 2 pairs in each group working. IL descriptions collected
10 minutes	post-training analyses of "Mailtool" example	post-training analyses of "Windows" example	post-training reports
15 minutes	debriefing & feedback	debriefing & feedback	videos of 2 pairs in each group (those not videoed earlier) being debriefed. Completed feedback forms

Table 1: training timetable and data collected

The training examples were designed to introduce students gradually to the important features of the Instruction Language. Students were introduced to the syntax of the language and to the idea of hand-simulating the user's changes in goals and knowledge. Discussion also covered the process of writing the IL and the choice of representation.

We also covered three important but subtle points: that the same device action can correspond to two or more different conceptual operations (i.e. that users can do the same thing for different reasons), that the same device object can correspond to information about different things, and that when there are alternative ways by which users can know something, they might select one rather than the other. In particular, users might sometimes incorrectly predict ("track") the effects of actions and assume that the action has the predicted effect without explicitly checking it. This can be important if there are aspects of the design that users might predict incorrectly, as discussed below in the context of the Mailtool example.

2.1 THE TEST PROBLEMS

The test problems were chosen to strike a balance between ease of writing an IL description and likelihood of that analysis yielding insights into the usability of the device. In the hands of an experienced analyst, most of the insights into usability difficulties with a device come from the exercise of trying to write an IL and failing to do it in a complete and consistent way. Novices would be unable to distinguish between difficulties that were due to their inexperience

and difficulties that were due to real design problems. Since the main question being addressed in this research was how effectively students could write IL descriptions after one day's training, the devices selected for analysis were mature, consistent and usable enough for novice IL writers to be able to write an IL description.

The "Windows" example

The "Windows" example involves describing the task of resizing a window on the Macintosh so that a complete picture can be seen, and then returning it to its original size (Figure 3). Students were told how windows are moved and resized on the Macintosh. They were asked to consider a scenario in which someone is comparing the contents of the two windows and wants to temporarily view the whole of a big picture in the lower window by re-sizing that window to about three-quarters of the screen height, and then restoring it to its present state.

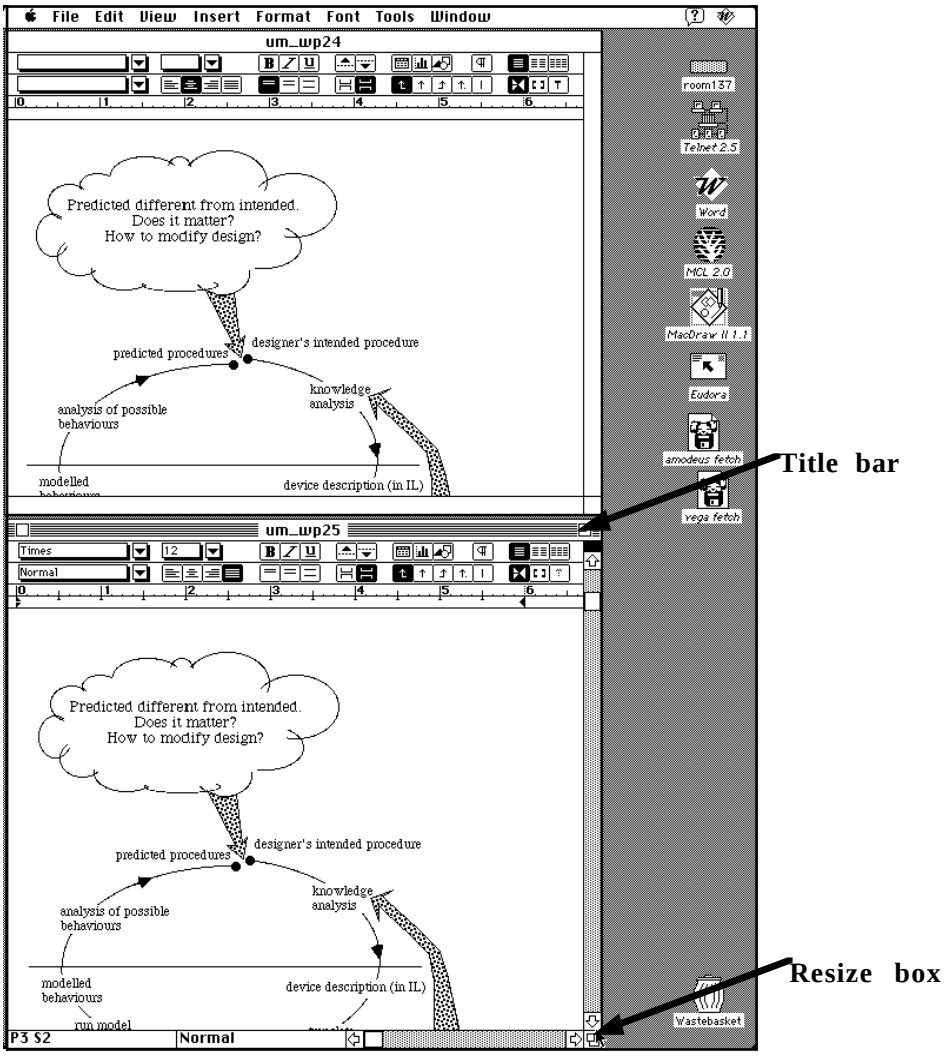

Figure 3: "Windows" example

For this scenario, the window can only be enlarged if it is moved first, and it must then be made smaller before being returned to its original position (because otherwise the resize box disappears off the bottom of the screen). Our model answer includes three conceptual operations: to resize the window, to move it so that it can be resized, and to move it to get it to a particular location on the screen (see Appendix). In this case, all the information users need is available to them from the screen, so they do not need to track any information. However, the "task" consists of two tasks, to be achieved in sequence. For the first (enlarging the window), the user needs to move the window before it can be resized; the order constraint is imposed by the device. For the second (reducing its size), the user must resize the window before moving

it, otherwise the resize box disappears off the bottom of the screen; as the order is not constrained by the device, users may make order errors.

The "Mailtool" example

The "Mailtool" example involves describing the task of sorting mail messages from two folders into a third one. At any time, some folder has been "loaded" into the tool, and the headers of the messages it contains are displayed in a scrollable list. The user can load a folder of messages and can move a message from the currently loaded folder to some other folder. The tool has a slot named *Folder* which can hold the name of a single folder (see Figure 4). Students were told how a user can load a folder, move a message, and highlight a particular message.

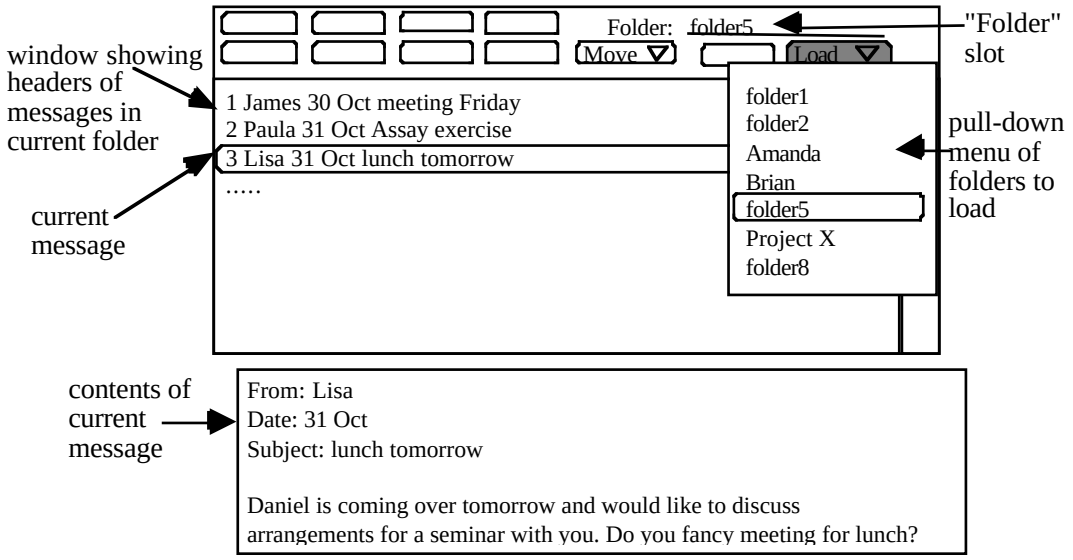

Figure 4: example "Mailtool" screen

Students were asked to analyse a scenario in which a user was sorting messages, selecting some of the messages from two different folders and moving them to a third folder.

Students writing IL descriptions of this scenario were prompted to consider two cases: where the user looks at the screen to maintain knowledge of the state of the device, and where the user mentally tracks the effects of some operations without checking the display.

This task is more complex than the Windows one. Firstly, there are two ways to achieve loading a folder and moving a message ("quickly", if the name of the required folder is already shown in the "folder" slot, and "slowly", explicitly naming the folder, otherwise); the user would select these operations in different circumstances, which should be specified in the IL. Secondly, there is one location on the display that contains the name of a folder; this might be the folder that is currently loaded, *or* the one to which messages are being moved. The user has to track which folder is currently loaded, has to track which folder messages are moved to, but can look at the screen to see which folder name is displayed on the screen. This is a case where one display item serves two purposes. Thirdly, if the user does not rely on the screen for information, but tracks the names of the "from" and "to" folders then she may make errors by failing to note that the "to" folder name has changed when she only intended to change the name of the "from" folder. A detailed analysis of this example is presented by Blandford & Young (1996).

2.2 THE DATA COLLECTED

To address the main question, how effectively students could be taught to write IL descriptions after a day's training, the data collected were:

- the IL descriptions of the Mailtool and Windows examples.
- video recordings of half of the pairs (2 for each example) constructing their IL descriptions. The pairs were those who volunteered; the main consideration was how comfortable they, as non-native English speakers, felt about working entirely in English for this session.
- video debriefings of the remaining pairs, in which they were asked to talk through how they had constructed their IL descriptions.
- feedback forms on which the students were asked to give their views of the qualities of the training, and which concepts they found easy or hard to grasp.

To assess the subsidiary question, whether using the IL makes a discernible difference to the kinds of usability assessments the students make, the main sources of information were the pre- and post-training written usability assessments, which the students wrote individually. Video data was also used to establish a more detailed understanding of how the writing of the IL influenced the assessments.

Finally, in order to assess whether other individual factors might have influenced the results, students were asked to provide brief details of their academic backgrounds and prior experience with computer systems of the types used in the test examples.

2.3 METHODS OF ANALYSIS

The tutors devised a marking scheme for the IL descriptions by agreeing a set of criteria on which to mark, weighting those criteria to reflect their relative importance (in the views of the tutors), then allocating a mark to each IL description using a scale of 0-4 for each criterion. Criteria covered both appropriate use of syntax and broader concerns such as consistency, completeness and accuracy of the description. The tutors independently assessed the IL descriptions produced both quantitatively and qualitatively, identifying omissions, misconceptions, inappropriate use of the syntax, etc. The assessment made just on the basis of the students' written output was verified against the corresponding video data and also information from the students' feedback forms.

The pre- and post-training usability assessments written by the students were analysed in terms of all the statements made by the students. These were assigned to one of three categories: design recommendations (suggesting how the design might be modified to overcome any problems identified); usability issues that would not emerge as a result of doing an IL analysis; and usability issues that could emerge as a result of doing an IL analysis. The allocation of statements to categories was verified by the second tutor assigning statements to the three categories blind.

Points of contrast between the pre-training and post-training reports were established. As explained earlier, it was not anticipated that students would gain substantial design insights as a result of writing IL descriptions, because the designs selected for analysis were chosen to be clear and consistent enough for students to be able, in principle, to write IL descriptions of them; therefore, they had few outstanding knowledge-based usability problems. However, we wished to compare the types of usability points raised in the pre- and post-training in terms of both content and the way the issues were expressed.

The video data was also analysed. The framework in Figure 1 motivated the video analysis, focusing our attention on phenomena that the framework's first two modelling links highlight as underpinning the modelling process: the skills required to translate the design problems into the IL notation, and evidence of whether or not IL helped students to understand the problems. These provided the initial categories to begin classifying segments of the verbal transcripts produced from the videos (as presented in Section 3.3). Additional categories were created during analysis to reflect emergent themes, as discussed in Section 4.

3. Results

In our presentation of results, we aim to draw out points of general interest, in relation to training practitioners in the application of semi-formal user-centred techniques, rather than points that are specific to the IL.

In the following sections, direct quotations from students' reports and video protocols are used to illustrate particular points. The students are labelled according to pairs ("Mailtool" pairs are numbered 1-4, and "Windows" pairs are 5-8), with individuals in each pair labelled A and B.

3.1 HOW WELL DID STUDENTS LEARN TO WRITE IL?

To address this question, we looked at the marks allocated to the students' IL descriptions, and also the tutors' qualitative assessments of those descriptions. The 'bottom line' quantitative marks for the students' IL descriptions ranged from 52% to 89%, with the average being 71%. Obviously, there are no strong claims that can be made about this figure, beyond it being some measure that most students wrote syntactically good IL descriptions and were able to grasp the main principles of the IL.

The real interest comes in the aspects of the IL that students consistently failed to attend to, or consistently misused (e.g. using categories inappropriately):

- students produced high standard descriptions of objects, functions/predicates and conceptual operations. They showed a good understanding of basic concepts.
- many students simply omitted several other categories, such as defining initial states, tasks or device effects, from their descriptions.
- most students did not specify in detail what users would know and how, or what the effects of an operation would be and how the device state would change. In some instances, students failed to make an adequate distinction between the user description and the device description.
- similarly, aspects of the IL description that depended on the students hand-simulating the modelled user's cognition were not well attended to. So, for example, there were several instances where students specified preconditions that could not be satisfied.

In summary, students were able to produce IL descriptions that were largely syntactically correct but they omitted aspects of the problem that are described primarily to support hand-simulation.

3.2 DOES USING THE IL MAKE A DISCERNIBLE DIFFERENCE TO THE KINDS OF USABILITY ASSESSMENTS THE STUDENTS MAKE?

As already discussed, the test material was such that users were not going to identify major usability problems as a result of writing IL descriptions. Also, the students had already received a good basic training in usability analysis, as part of their course on HCI. Consequently, the differences between the pre- and post- training analyses were fairly subtle; the main point of interest was how the analyses differed qualitatively.

Windows

Most pairs working on this problem did not complete their IL descriptions. Process data of pairs working shows that the main reason is that they had difficulties deciding how to represent window locations and sizes, and how to describe the user's task.

Most pairs focused on the enlarging task (for which the usability difficulties are very minor) and paid less attention to the second (making the window smaller) task, for which we might have expected more of them to spot the potential order error (of moving the window before resizing it). Some pairs *only* considered the first task, so were unlikely to spot aspects of the design that were pertinent only to the second.

Most subjects, in both the pre-training and the post-training, outlined the sequence of actions the user would have to perform to achieve the tasks. These descriptions varied in their user-centredness, in terms of the reasoning they gave, or implied, for the assertion. At one end we have the prescriptive:

> *[S.1A, pre-training]: I don't really see the big problem. You can solve it by 4 to 5 operations on the lower window:*
> *#1: move lower window in upper direction*
> *#2: resize lower window*
> *#3: resize lower window to a size smaller than its original size*
> *#4: move the lower window back to its original spot*
> *#5: resize the lower window so its bottom side is back at the bottom side of the screen.*

At the other, we have statements such as:

> *[S.8A, post-training]: A window cannot be resized any further than the bottom of the window. When the user is resizing and she hits the bottom of the screen, the window can no longer be resized. When she wants the window to be bigger, she has to move the window up first and then continue the resizing.*

For this example, both pre- and post-training subjects were in a position to identify the main problems, as shown in Table 2. The main points of contrast were as follows:

- while all of the pre-training reports included design recommendations (of ways to modify the design to relieve the problem), fewer of the post-training reports included such recommendations.
- the pre-training students made a wide range of usability observations. Of these, 8 were points that we might reasonably have expected to emerge as a result of doing an IL analysis, and the other 6 were usability points that were not IL related. So, for example, general comments about sequences of fewer actions being easier to learn, comments that relate to screen layout, and speculations about the mouse not working would not emerge directly from an IL analysis, whereas comments about order errors and the user knowing how big to make the window can be related directly to an IL analysis.
- the pre-training subjects achieved a better balance than the post-training subjects in considering the two sub-tasks (make the window bigger, then make it smaller). (In table 2 the first 4 of the IL-related usability points relate to the first task, and the remaining 4 to the second; in the pre-training, 9 comments relate to the first task and 11 to the second; in the post-training the split is 13:8). Evidence from the IL descriptions and the video data suggests that this is because most of the post-training subjects focused their attention on the first task, and did not consider the second properly.
- the two IL-related usability points made by most of the pre-training subjects referred to the sequence of actions the user would have to perform to achieve the task. Post-training subjects are less prescriptive.

WINDOWS	pre-training subjects								post-training subjects							
Design recommendations	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B	8A	8B
hide tool-bars / add zoom facility or scrollbar to compensate for small screen size			x	x								x	x			
you could 'lock' functionality, so that both windows automatically scroll together			x	x												
there should be multiple resize buttons / move-bars	x	x	x	x	x	x	x	x			x	x				x
the resize operation should continue if you reach the bottom of the screen	x															
don't allow window to be partly off-screen			total: 14							total: 8					x	
there should be a function to go back to previous position and size	x													x		x
non-IL-relevant usability issues																
One window may obscure another					x			x	x				x	x		
It might be confusing that the user can click/drag anywhere on the title bar to move the window, but specifically has to click the bottom right hand corner to resize						x										
If the user cannot resize a window because it would otherwise fall off the screen, the user might think the mouse isn't working or that there is something wrong with the program			total: 10							x		total: 5				
It is not user-friendly to expect users to do two steps to achieve one goal	x		x	x		x										
sequences of fewer actions are easier to learn						x										
the user has to know the possible operations / make up a strategy	x				x				x							
IL-relevant usability issues																
When making the window bigger, the user must move then resize	x	x	x	x	x	x	x	x	x		x	x			x	
The user does not know how big the window should be for the picture to fit in				x							x	x				
window cannot be enlarged if size box is at bottom of screen										x	x		x	x	x	x
The user might not understand why she needs to move the lower window before resizing it									x							
If the resize box is off the screen, the user must move the window before resizing / A window can only be resized if the resize button is visible			total: 20				x			total: 21				x	x	
To restore the window to its present state the user must resize the window then move it back (then maybe adjust the size again)	x	x			x	x	x	x	x					x		
The user cannot properly define the desired size before moving it back		x					x	x	x					x		x
Order error: user might try to move title bar to old position and resize it to fit screen, but this is impossible because the resize box then falls off the screen							x		x							

Table 2: usability points made by subjects in pre- and post-training reports on the Windows example

In summary, this task was one where it was easy to spot the difficulties. These students had a good general HCI training prior to the IL training day so, not surprisingly, they could all describe the problem. There is a trend away from prescription (on the user — "the user must do this then this then this to achieve the goals" — and on the device — "the way to fix this is to have multiple resize boxes") towards an analysis based on the user's knowledge (e.g. "The user does not know…" or "The user might not understand…").

Mailtool

All pairs working on the Mailtool example produced descriptions that were syntactically of a high standard. However, all included two functions that corresponded to `loaded-folder()=folder` and `selected-folder()=folder`. That is, rather than distinguishing folders in a way that was relevant to the user's tasks (i.e. which folder messages are being moved *from* and which they are being moved *to*), they made a more device-oriented

distinction (which folder is loaded, and which appears in the "folder" slot). This made it less likely that any of them would identify one particular usability problem – that users might lose track of which folder was loaded and which messages were being moved to, and hence move some messages to the wrong place.

MAILTOOL	pre-training subjects								post-training subjects							
Design recommendations	5A	5B	6A	6B	7A	7B	8A	8B	1A	1B	2A	2B	3A	3B	4A	4B
allow user to move multiple messages at one time	x	x	x	x							x		x			
it should be possible to delete / copy / edit / scroll through messages	x	x			x	x										
it should be possible to create a new folder / duplicate a folder	x	x														
better message headers					x	x										
introduce another folder slot			x	x	x		x	x	x	x					x	
clear folder slot when you change function / always have to specify destination (no quick move)							x				x		x	x		
use drag-and-drop to move messages								x							x	
non-IL-relevant usability issues																
A lot of nasty things happen when you can't find a certain message														x	x	x
The operation of the buttons is somewhat confusing					x			x			x				x	x
You don't know if the folder you want to move to is going to be loaded							x	x								
It's easy to move multiple message to one (other) folder									x							
IL-relevant usability issues																
The 'folder' slot does not always display the name of the folder of which the messages are shown. / The user has to track which folder is open					x	x	x	x	x	x	x	x			x	x
If you click on the MOVE button while a folder name is already in the folder slot, the message will be moved to the folder it is already in.	x		x	x								x				
It might be confusing that there are two ways of loading and/or moving.								x			x	x				
In the task there was only one folder to which you had to move messages. So in this task there wasn't a problem of tracking the slot value.													x		x	
if you had to move messages to two files then this default thing would cause problems.																
the user might lose track of which folder is the move-to folder													x	x	x	x
before you can select a message, the folder it's in has to be open															x	x

total: 19

total: 9

total: 4

total: 7

total: 8

total: 18

Table 3: usability points made by subjects in pre- and post-training reports on the Mailtool example

Identification of this problem depends on recognising that the user might rely on tracking information that is actually available on the screen, with the qualifying statement that if the user always relies on the screen she should not experience problems. Some of the students recognised that there might be a problem in this area (see Table 3), but did not articulate it very clearly. The main points of contrast are as follows:

- while all of the pre-training reports included design recommendations (suggesting ways to modify the design to relieve identified problems), the post-training reports included fewer such recommendations.
- although many of the pre-training subjects suggested introducing another folder slot, the post-training subjects stated more clearly why this is required.
- the post-training students made IL-related usability points with higher frequency than pre-training subjects.

In summary, there is, as noted above for the windows example, evidence of a move towards a more user-centred way of expressing their insights.

Statistical analysis

Table 4 summarises the numbers of different points made by subjects in their pre- and post-training reports on the two examples.

	Windows		Mailtool	
	pre-	post-test	pre-	post-test
Design recommendations	14	8	19	9
Non-IL issues	10	5	4	7
IL-related issues	20	21	8	18
TOTAL	44	34	31	34

Table 4: number of points made by subjects in their reports.

Linear modelling of the data shows that there is no main effect of problem type (Windows vs. Mailtool), nor of training (pre-test vs. post-test), nor of subject group (Windows first vs. Mailtool first), thereby indicating that there are no unwanted differences between groups or problems, and that there is no overall effect of the training on the total number of comments made by the students. However, there is a significant effect ($p < .05$) of the interaction between category type (design recommendation vs. IL-related usability point) and training, indicating that the training does seem to have an effect on the kind of comments that students make.

The interaction was examined in more detail by means of chi-squared tests applied to the data for the Windows and Mailtool examples separately. As is apparent from inspection of table 4, the tests confirm that for the Mailtool example there is a significant tendency ($p < .02$) for the students to shift from making design recommendations before the training to making IL-related usability points after the training. For the Windows example (table 2), there is a slight shift in the same direction, but it is not significant ($p \approx .5$).

Given that students had just received intensive exposure to the IL, it is hardly surprising that their post-test results show a shift away from design recommendations to IL-informed usability insights. In some cases this might be seen as a negative asset (new design solutions being of more value than a deeper understanding of a problem — particularly when that understanding depends on time-consuming analysis). There are many cases in which the costs of analysis outweigh the benefits, but there are also cases (e.g. Bellotti, Blandford, Duke, MacLean, May & Nigay, 1996) in which a deeper understanding of the problem can help the designer generate workable solutions to it. PUM analysis does not replace good craft skill, but is a technique that can be invoked when appropriate. This study did not address the utility of modelling, but was concerned with the development of modelling skill; this result indicates that such a skill is being developed.

The views of students

As well as conclusions we can draw from the data, we also have the views of the students on the role of the IL in informing their usability assessments. They were asked: "In the exercises, to what extent did using the IL give you insights into the interface design problems, or help you to spot the problems more easily?"

Seven students wrote comments similar to the following three:

Anon: *helpful - keep in mind that users are liable not to track effects*

S.3B: *... Without the IL, I'd probably say too easily "just move this file" without thinking about subgoals.*

S.4B: the main useful aspect was the process you had to do to model the situation in IL. While trying to model the situation, many design problems will be found just by doing the modelling. Group discussions often went like this: try to model —> hey this is a problem —> solution could be —> OK let's go back to modelling.

Although the framework in Section 1 was not presented to students, this last comment very clearly describes the links presented there (iterating from design representation to problem insights and design recommendations). This indicates that this student's view of the role of modelling matches that presented in Section 1.

Three students were less positive about the usefulness of IL. Their comments included:

S.6A: IL seemed more a way of modelling a problem after you identified one

S.6B: specifying everything is problematic; tracking information and filters sometimes produced useful insight. Is it worth the trouble for the designer??

This suggests a general perception that the IL may have some benefit, in encouraging people to consider user knowledge and user behaviour in more detail than they would do otherwise. However, after such a short exposure to the technique, the assessments are inconclusive.

Summary

Overall, the main points of contrast between the pre- and post-training results were not in the number of usability points raised, but in the justifications given for them. For example in the pre-training we have:

S.1B [Windows]: Problems: to make the lower window larger, the user had to move the window upwards. When restoring the window it's impossible for the user to get the right size, so the user makes it smaller than its original size, places it back and then re-sizes again.

S.6B [re. folder slot in Mailtool]: How did I diagnose this? Two operators have side effects on the same variable: bad computer science karma.

This can be contrasted with some of the justifications given in the post-training, such as:

S.5A [Windows]: The task is essentially about resizing the lower window. The user might not understand why she should need to move it first. This was the main problem in writing out the IL: to make explicit the relation between resizing and moving.

S.2B [Mailtool]: The user has to track which folder is open. After you have moved a message the slot contains the name of the destinations slot. And you've lost the name of the "open" folder.

Whereas the pre-training justifications tended to refer to general principles of good design, or to the action sequence a user would have to perform, post-training justifications were more likely to refer to how the user would know things, or misconceptions the user might have.

3.3 HOW DO TRAINEE SOFTWARE ENGINEERS USE IL AFTER ONE DAY'S TRAINING?

In this section, we look at the process of using IL to tackle design problems, drawing mainly on process data captured on video. How did students reason with the formalism when analysing the problem? Did using the notation draw attention to relevant issues, or provide insight into user cognition? Although the present data are drawn from novice performance, such questions are, of course, critical for assessing the longer term potential of IL as a tool for reasoning in real design. The video data was gathered to provide insight into the *process* of IL-use (as opposed to analysing just the *product* at the end of the session).

Characterising the overall process of using IL

One aspect of the cognitive load on IL beginners is remembering what has to be addressed for an IL to be complete in a grammatical sense; clearly it can be used in a more or less effective

way. The students were provided with a single sheet summary of IL and a list of heuristics giving guidance on how to write it. The summary sheet was observed to be a valuable template for them when they came to construct their own IL specifications, cueing them to the elements they had to consider in order to get going. This template was followed in a loosely serial manner, starting with *Objects*, then *Functions*, followed by *Predicates*, and then the *Operations*. In the videoed exercise, students were observed to memorise the structure at the level of *Operations*, which have a micro-structure of slots to be filled (*user-purpose, precondition, filter, tracked-information, device-command*); some pairs learned to write down this template from memory as soon as they began to specify a new operation.

The overall IL authoring process is best described as loosely serial. Students declared some obvious Objects, Predicates and Functions based on the problem scenario description, then revised, deleted and added to these as they reasoned in more detail about specific operations. The iterative nature of IL construction had also been stressed repeatedly in the teaching and class exercises. This is consistent with studies on design and programming cognition (Siddiqui, 1985; Guindon, 1990; Visser, 1990) which have reported that opportunistic strategies are overlaid on textbook top-down design and programming methodologies.

Another well-documented approach to design reasoning is that of reasoning about specific scenarios of system or user behaviour (Young and Barnard, 1987; Karat & Karat, 1992; Carroll, 1995). Scenario-based reasoning is, in a sense, embedded in IL. For instance, walking through a user task sheds light on how Objects, Predicates and Functions should be (re)expressed from a user point of view. Similarly, in order to specify the *user-purpose* or what information is *tracked* for a particular operation, one must reason explicitly about user cognition by imagining the user performing the task in question—why are they doing this, and what information must they keep track of? Developing an IL specification encourages software engineers to construct user-centred scenarios.

Reasoning with IL constructs

One important role for any notation is to alert analysts to particular issues. The students' use of IL could be observed in the video transcripts, as in the following examples.

Reasoning about user-purpose

In the following example (taken from the Windows problem), the requirement to specify the *user-purpose* for the *resize-window* operation leads the subjects to an insight about enlarging and reducing windows—that the user purposes of doing so, and hence the criteria for judging whether a window is small or large enough, are very different.

Pair 7 - Windows:

[they discuss: do we need to specify that the user-purpose of resizing is to enlarge or to reduce?]

S.7B *The user-purpose of enlarging is to see the contents of that window, and the user-purpose of reducing is to make another window visible*

S.7A *becomes very complicated - two resize functions - confusing.*

S.7B *resizing one window to see another window - that's complicated. First let's try enlarge.*

[First operation "resize-window" is changed to "enlarge-window"]

S.7A [gesturing on screen shot of windows] *So to enlarge you click on resize box and drag away* [gestures dragging down and to the right]

S.7B *or drag down*

S.7A *and then it becomes bigger*

S.7B *now I see the importance of moving the window - you cannot resize this* [lower] *one*

S.7A *yes that's what I meant - you can't resize that*

This extract also illustrates how the need for *two* functions to express resizing highlighted a possible source of user confusion, and how walking through the task highlighted the fact that the design of Macintosh windows means that the lower one cannot be resized without first moving it.

Reasoning about *tracked information*

At one point, Pair 4 discussed *tracking* for all the operations. They returned to this issue 23 minutes later when considering a more expert user:

Pair 4 - Mailtool:

S.4A *so you don't see what is selected - I don't see the problem - you're already tracking things, but not the selected folder... we said it was a side-effect*

S.4B [finds their IL, and notes that the *move-to-folder* operation has an entry for tracking the active folder and that the selected folder side-effect is visually detected. S.4B notes that in the new situation]—
The user can see it, but won't - he tracks it now - he remembers it.

There are three features of interest here. Firstly, it is a good example of how an IL can be used as a cognitive resource to assist reasoning about the design. Initially, the `tracked` slot in the `Operations` schema provides a checklist of things the designer must consider; while one could simply tell designers to "think about what the user must bear in mind", the more structured IL representation seems to be a richer resource in this respect. The students co-ordinate their discussion by moving systematically around the IL from one operation to another.

Secondly, in the above extract, the IL acts as a resource to help answer a 'what if' question: what if the user does not look at the screen? The students are able to interpret this new scenario relative to the current state of the IL, by going to the relevant IL element (`tracking`).

However, and this is the third point, this particular pair do not deduce that non-visual tracking in the new scenario may lead to problems (the user may fail to track which folder is active). What this demonstrates is appropriate internal use of IL (within the closed world of IL), but failure to identify the external problem.

Reasoning about *preconditions*

In IL, before a particular operation can address its user-purpose, its *preconditions* must be satisfied. For the analyst using IL, this is an opportunity to assess whether the user can reasonably be expected to know what these are, and how to satisfy them.

In the extract below, Pair 6 explain the problems which they anticipate for the user in terms of *preconditions* to operations which must be satisfied, and also with respect to the information which must be mentally *tracked*.

Pair 6 - Windows debriefing

S.6A *at the end we had some problems with locations and sizes.*
we had 2 operations for move-window ... and we had resize-window to size:S

S.6B *and somehow we wanted to relate these two, because it was similar to the other problem where in order to resize the window you have to move it, but how is the user going to know this?*

S.6A *so we didn't know how to express that there was no room to resize the window*

...

S.6B *so what we asked in the end was: is it sensible for the user to know the preconditions and the tracking information and everything, and we decided no, it's not sensible to require the user to know where he's going to locate the window in order to resize it.*

The IL has given the students a vocabulary for reasoning about how users know things about the device, and where that knowledge comes from.

4. Discussion

As stated in the introduction, the aims of the work reported here have been relatively modest: to establish how trainee software engineers use the "PUM Instruction Language" (IL) after a day's training. We have explored the use of different types of data collection in helping us assess the level of understanding students acquire in this time. In the course of this, we have also used the study to develop a better understanding of the nature of expertise in using IL. We now re-visit issues raised earlier about the role of formality, the centrality of the underlying theory, and the nature of craft skill.

4.1 THE ROLE OF FORMALITY

As discussed above, a grounding in the theory and systematic application of a particular formalism equips a designer with a conceptual framework with which to examine problems, which can then lead into more rigorous application of the formalism if the problem is particularly complex. For example, using IL sensitises one to 'tracked information', and in cases where tracking seems to be significant the analyst can do a fuller investigation to establish what the consequences of users tracking information might be. As discussed in Section 3.3, students were observed to be using the notation as a resource to support their reasoning in this way.

As expert analysts, the tutors would not normally write a full IL description of all aspects of a substantial design; rather, they would identify important aspects of the design, generate interesting candidate scenarios, and sketch out important bits of IL description at a suitable level of detail—from fairly informal as a rough tool for reasoning, to a complete portion of IL code generating a running model. Novices do not have the experience to enable them to do this, or might do it inappropriately. For example, Pair 4 introduced a new category *side-effect*, which expressed something they wanted to say (that an action has a side-effect, which the user needs to be aware of), but also served to obscure some difficulties that users might experience.

Another factor that determines the rigour with which a formalism is used is its role as a vehicle for communication between designers. The video data shows several incidents where partners initially used the same terms to mean different things; it was as they moved from general discussion to IL description that they recognised and resolved these ambiguities and reached a common understanding of the problem. In this sense, the formality served to reduce ambiguities in their communication with each other.

Although there were no overall measurable differences in performance that could be accounted for by students' different backgrounds, one of the psychology students commented, unprompted, that a semiformal language was a good way to bridge between computer scientists and psychologists, since it gave them a vocabulary they could both work with:

S.2A: maybe because of being a psychology student I'm not (very) used to using this formal language (it looks a bit like programming to me!) …
It's a very nice way to communicate with other people (especially computer science students) since it's very structured

In contrast, some of the computer scientists noted on their feedback forms that they would have preferred a more formal presentation of the syntax of the IL:

S.5B: [IL could perhaps be] more formal so you know what commands etc. may be used.

An understanding of how knowledge is used within a running model provides the most useful guide to what should be expressed in the IL (as the analyst envisions how the user

knowledge would affect the behaviour of a running model), but this understanding is not available to novices.

There is a tension between the requirement to allow the language to be used flexibly as a tool to aid the analyst's thinking, and the requirement to gain leverage on the problem by exploiting the constraints of the language and the underlying theory. However, it is clearly not possible for students to appreciate this difference, or acquire deep theoretical knowledge, in a matter of hours.

4.2 CENTRALITY OF THE UNDERLYING THEORY

As discussed by Blandford and Young (1996), there are some issues for which the analysis can be done simply in terms of users' knowledge, and others for which various depths of understanding of the underlying cognitive theory is necessary. The two test examples differ significantly in terms of the role played by the underlying theory in guiding the analyst towards insights.

For the Windows example, the main insights are gained through the experience of describing the task scenario in terms of the Instruction Language. For example, by specifying that having the resize box visible is a precondition for being able to resize the window, and therefore having to specify an operation whose purpose is to make the box visible, the analyst has to make an explicit statement that there are different reasons for moving a window, and that if the window has to be both moved and re-sized there might be order constraints on these two operations. Most students who specified such operations in the IL were able to draw the appropriate inferences about the real world consequences of their IL descriptions.

In contrast, for Mailtool the main insights come from hand-simulating the modelled user's cognition. For example, the means by which the modelled user knows things (by looking or by tracking) is central to the analysis. If student analysts do not interpret their IL descriptions in terms of the consequences of different descriptions on the modelled user's knowledge, they cannot gain the main insights. Students made several usability points in the post-test reports — e.g. "the user might lose track of which folder is the move-to folder" — that reflect an IL-informed understanding of the problem, but omitted points that would have emerged from extended hand-simulation. The greater dependence of the Mailtool problem on model-based reasoning (as opposed to simply representing the problem in a particular notation) probably accounts for the greater shift from design recommendations to IL-related usability points when compared to the Windows problem (see Section 3.2).

The question of how much theory novice analysts need to understand before they can exploit a user modelling technique has been faced by the developers of other theory-based approaches; the answer which seems to be emerging in relation to PUM and the IL is that it depends on the issue that is being modelled. Profound insights are not cheap; the better an analyst's understanding of the underlying theory, the more likely it is that the modelling will achieve useful results. In terms of selecting training examples, the depth of theoretical understanding needed to identify usability difficulties is one important dimension on which to base selection.

4.3 IDENTIFYING AND ENCAPSULATING CRAFT SKILL

Whilst HCI design formalisms offer the designer more rigorous ways to represent and reason about a design, their effective application often assumes a significant degree of informal craft skill. All too often, papers focus on the relationship between a design formalism and the domain to be represented, stopping short of addressing the crucial issue of its use in practice: are designers able to use the formalism effectively by making appropriate connections between the formal description and the domain implications (Feather, 1993)? Formalisms cannot be meaningfully evaluated if their application in practice is ignored. There are substantial, hidden skills involved in making a design formalism practically useful, and for a variety of reasons (e.g. deriving from the traditional engineering and science cultures) these skills often remain unarticulated (Bowers, 1991; Subrahmanian, Konda, Levy, Reich, Westerberg & Monarch, 1993).

As is widely recognised, the development of training materials can help experts to identify some aspects of their skill; the requirement to produce heuristics, templates and examples can all help the expert to articulate essential components of their skill. However, in practice, there was little evidence in the video protocols of students referring to the list of IL heuristics. It was much more common for them to refer to the template that specified the IL syntax, or to one of the training examples, to check how the IL had actually been used there. This suggests that while there is a value in making heuristics explicit, particularly as a tool for the trainer, they are less useful as a resource for students than the template that summarises the notation and also other examples, from which they extrapolate. This, in turn, suggests that the role of examples in encapsulating and communicating craft skill needs to be further developed.

There are certain aspects of craft skill that were found to cause particular difficulties in the study. The first is reasoning explicitly with the cognitive theory (e.g. in hand-simulation) and using an understanding of the theory to determine whether a particular notational relaxation is appropriate or not, as discussed above in relation to 'side-effects'. Two other difficulties that emerged were that of identifying an appropriate level of detail for description, and making a clear distinction between the device state and what the user knows about that state.

Specifying IL at the right level of detail

One of the hardest decisions when using any representational technique is to analyse at the right level of detail. Difficulties with this have been observed in studies of designers learning to use new design formalisms, such as argumentative design rationale (Buckingham Shum, MacLean, Bellotti & Hammond, 1997), a cognitive modelling expert system (Buckingham Shum & Hammond, 1994a) and a notation for describing the ontology of a system (Blandford & Green, 1997). This also proved to be a difficult issue for some of the students learning IL.

The excerpt below illustrates the issue of level of description with respect to whether *preconditions* need to include *device commands*:

Pair 1 - Mailtool

S.1B *should we have as a precondition that move is selected? And here* (indicates other operations) *that the relevant buttons are selected?*

[they check a training example to see how this was done]

S.1A *do preconditions include physical actions?...*

we'd have to have move mouse, press button, drag it... too low level— leave it!

The general answer to the question, "Do I need to specify X...?" is "It depends on what you're analysing". However, choosing the level of description is difficult for beginners because it requires considerable familiarity with the notation to know how it can be used in different ways. Of course, analysing at the right level is partly dependent on human factors skill independent of IL expertise. That being the case, however, it would still be useful for trainees to see examples of IL working at different levels, with the rationale laid out explicitly for the way in which it has been used. Armed with these different examples, they would then have a greater awareness of the representational options available to them.

Distinguishing between device state and user knowledge

One of the central tenets of the PUM approach (Young, Green and Simon, 1989) is that a language like IL should be sufficiently similar to a conventional high level programming or specification language for software engineering skills to transfer, with minor adjustment, to user-centred analysis. Preconditions and end-states are familiar concepts to computer scientists. However, this familiarity with languages for specifying device behaviour is a double-edged sword. Whilst it may enable designers to use their existing expertise to reason about users in a new way, it may also have the unfortunate side-effect of leading them back into their more familiar device-centred mode of thinking, because it is so similar to what they are used to doing.

Students had little difficulty in writing syntactically correct IL descriptions. For example, they made sure all variables were bound, and definitions of objects, functions and predicates did not present a problem. Thus they seemed able to exploit their general programming expertise in writing IL descriptions. However, transfer of this expertise is not without its problems.

For example, for Mailtool, all pairs of students defined just one `selected-folder()=folder`. Some of them also defined a `loaded-folder()=folder`. This identifies folders in terms of which one is loaded and which one is displayed in the "folder" slot—i.e. still reflects the device representation. This contrasts with a task-oriented description, using `selected-from` and `selected-to` folders, the task being to "move messages *from* this folder *to* that one". The fact that students split the world up in a subtly different way accounts for the fact that none of them clearly articulated what we had identified as the main problem with this device.

Their computer science background appeared to influence one pair's choice of representation more extremely, leading them to describe an ideal device that did not match the description they were given. Reference to the video data for this pair suggests that the main reason for their producing an inaccurate description was that they were thinking of messages and folders as being similar, so they provided similar operations (`select-message` and `select-folder`) to manipulate them.

S.1A: *Do we need an operation "select message"? — because we don't have an operation "select folder"*

This would appear to be motivated by a general HCI principle, and good design practice, of being consistent, but fails to describe the essential features of this device in this situation.

Students were clearly aware of this difficulty, and several expressed the view that distinguishing between the device state and the user's knowledge of that state was:

S.7A: *quite hard, because most of us were looking at the problem from an implementer's point of view*

Making the conceptual shift required to adopt a user-centred perspective is not something which IL as a notation on its own can enforce, but which it can encourage. The examples above illustrate both advantages and dangers of IL's intentional similarity to computer science languages.

4.4 A REVISED FRAMEWORK FOR UNDERSTANDING THE ROLE OF MODELLING IN DESIGN EVALUATION

In Section 1 we presented a framework for understanding model-based usability analysis. We can relate the results of this study back to that framework. Figure 5 is a revised version of Figure 1, adding further links to support the discussion here. In particular, we have divided Link 2 to include an explicit step of model-based reasoning (applying the underlying cognitive theory) and added Link 0, which represents craft-based insights.

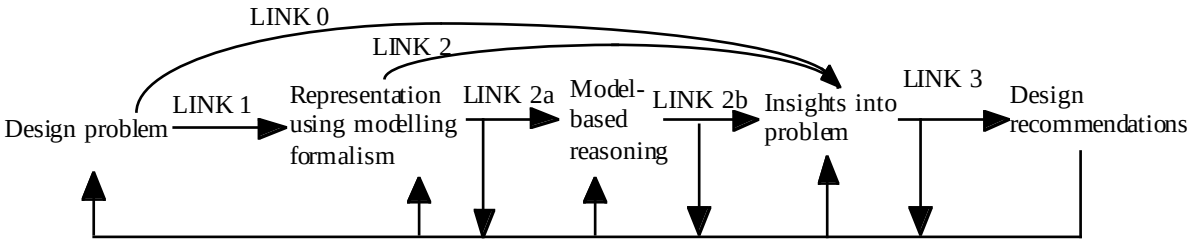

Figure 5: Links to negotiate in a usability analysis

The pre-training usability reports (Section 3.2) can be characterised as showing students traversing Links 0 and 3, using craft skill to identify various usability issues ("insights") and propose design solutions. In contrast, the post-training reports, which followed the generation

of IL descriptions (Link 1) focused more on expressing insights and the rationale for them (relating them to the modelling representation), with fewer design recommendations. Traversal of Link 3 was not emphasised in the training.

We can look at links 1 and 2 and assess how easy it was for the novices to traverse them. For the students working on the Windows example, there were clear difficulties in traversing Link 1, because of their difficulty in finding an appropriate representation of the design problem, but when an appropriate representation was found, traversal of Link 2 was unproblematic. Traversing Link 2 depended largely on exploiting the notational affordances of the IL.

The students working on the Mailtool example had little difficulty with Link 1 or with Link 2 — with using the surface features of the representation to yield modelling insights. However, the aspects of the problem that demanded more explicit model-based reasoning (such as hand-simulation) gave difficulties. This is largely because using the representation to gain insights actually involved applying the PUM cognitive theory in a way that they, as novices, were not yet equipped to do. This is an example of a design problem for which it is more appropriate to split Link 2 into two components (Links 2A and 2B) to describe the role of the cognitive theory in supporting reasoning.

5. Conclusions

Design and evaluation formalisms need to be usable. It is the responsibility of those developing such formalisms to understand and encapsulate the process skills required to apply them effectively in design. It can be as hard to articulate these skills as it is for a programmer to appreciate how a new system will be used by a beginner.

We have presented a case study of training software engineering students to use the PUM Instruction Language, and evaluated the degree of IL expertise acquired in the course of one day's training.

Results indicate that 6 hours was long enough for students to gain a good understanding of the surface syntax of the IL and to relate their descriptions to usability issues, but not to comprehend the significance of their IL descriptions in cases that depend on a deeper understanding of the theoretical model (requiring hand-simulation).

As is widely recognised, the process of developing and delivering training material forces one to reflect upon and encode methodological and other informal process skills (such as heuristics) which are regularly deployed in using the formalism. We have also shown how empirical studies of the approach in use can expose aspects of the approach which were assumed on the part of the expert, but were not clearly communicated. For example, the different demands imposed by different design problems had not been clearly recognised. Video data also showed how the formal representation could help students reach a common understanding of the problem (serving as a vehicle for communication between students). Another issue highlighted by analysis of the data is the danger of relaxing the formalism in inappropriate ways. While it may be tempting to present a technique as being flexible and informal, so that it is apparently easier to use, this can result in uncertainty about what the limits on that flexibility are, and in inappropriate use of the notation. By analogy with the discussion of tool use by Winograd and Flores (1986), a tool used by an expert can be used so seamlessly you hardly notice it; when it is in the hands of a novice you can see how it is operating, and what properties of the tool are achieving what.

This empirical study of trainee modellers has forced to the surface many of the implicit, tacit skills involved in turning a formalism into a practical tool for reasoning, and helped to identify many of the contributing skills in what otherwise remains a vague notion of 'expertise'.

Acknowledgements

It would not have been possible to conduct this study without the help and co-operation of a large number of people. We are most grateful to:

- Gerrit van der Veer for encouraging his students to take part in this study, for re-arranging their timetables to make it possible, and for his support in many ways while we were planning and conducting the study;
- Elly Lammers for making all the necessary administrative arrangements;
- the 16 students from the Free University of Amsterdam who participated with such professionalism in the study — even agreeing to talk only English for the duration! — and
- 8 students from the University of York and Manchester Metropolitan University who participated in, and gave us useful feedback on, a one-day pilot study that we ran to test the teaching materials and the experimental design.

We are also grateful to our AMODEUS colleagues for interesting discussions and stimulating conversations. This research was funded by the CEC under ESPRIT Basic Research Action 7040 (AMODEUS-2 Project) <www.mrc-apu.cam.ac.uk/amodeus>.

References

BELLOTTI, V., BLANDFORD, A., DUKE, D., MACLEAN, A., MAY, J. & NIGAY, L. (1996) Controlling Accessibility In Computer Mediated Communications: A Systematic Analysis Of The Design Space. *HCI Journal*. **11.4** pp.357-432.

BELLOTTI, V., BUCKINGHAM SHUM, S., MACLEAN, A. AND HAMMOND, N. (1995). Multidisciplinary Modelling In HCI Design...In Theory and In Practice. *Proc. ACM CHI'95: Human Factors in Computing Systems*, Denver, Colorado (May 7-11, 1995), 146-153, ACM Press: New York.

BLANDFORD, A. E. & GREEN, T.R.G. (1997) *OSM: an ontology-based approach to usability evaluation*. In Proc. International Workshop on Representations in Interactive Software Development, QMW. pp.82-91.

BLANDFORD, A. E. & YOUNG, R. M. (1993) *Developing Runnable User Models: Separating the Problem Solving Techniques from the Domain Knowledge.* In J. ALTY, D. DIAPER AND S. GUEST Eds. *People and Computers VIII*, Proceedings of HCI'93, Loughborough, 111-122, Cambridge University Press : Cambridge.

BLANDFORD, A. & YOUNG, R. (1995) *Separating User and Device Descriptions for Modelling Interactive Problem Solving*. In K. Nordby, P. Helmersen, D J Gilmore, and S Arnesen (eds.): *Human-Computer Interaction: Interact'95*. Chapman and Hall, 1995. pp. 91-96.

BLANDFORD, A. E. & YOUNG, R. M. (1996) Specifying user knowledge for the design of interactive systems. *Software Engineering Journal*. **11.6**, 323-333.

BOWERS, J. (1991). *The Politics of Formalism*. In M. Lea, Ed. *Contexts of Computer-Mediated Communication*, 232-261. Harvester Wheatsheaf

BUCKINGHAM SHUM, S., BLANDFORD, A., DUKE, D., GOOD, J., MAY, J., PATERNO', F. AND YOUNG, R. (1996). Multidisciplinary Modelling for User-Centred System Design: An Air-Traffic Control Case Study. In *People & Computers XI: Proc. of HCI'96, 11th BCS Conference on Human-Computer Interaction*, London (Aug. 20-23, 1996): Springer-Verlag, London.

BUCKINGHAM SHUM, S. & HAMMOND, N. (1994a). *Delivering HCI Modelling to Designers: A Framework and Case Study of Cognitive Modelling*. Interacting with Computers, 6 (3), 311-341.

BUCKINGHAM SHUM, S., MACLEAN, A., BELLOTTI, V. & HAMMOND, N. (1997). Graphical argumentation and design cognition. *Human-Computer Interaction*. 12.3, 267-300.

CARD, S.K., MORAN, T.P. & NEWELL, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates: Hillsdale, NJ.

CARROLL, J.M., Ed. (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development*. Wiley: New York.

FEATHER, M.S. (1993). *Requirements reconnoitring at the juncture of domain and instance.* In Finkelstein, A., Ed. *Proceedings of IEEE International Workshop on Requirements Engineering*, 73-76 New York: IEEE.

GUINDON, R. (1990). *Designing the Design Process: Exploiting Opportunistic Thoughts*. Human-Computer Interaction, 5 (2&3), 305-344.

JOHN, B. & KIERAS, D. (1996) The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on CHI.* , **3**, 320-351.

JOHN, B. E. & PACKER, H. (1995) Learning and using the Cognitive Walkthrough method: A case study approach. In *Proceedings of CHI'95.* pp.429-436. ACM Press: New York.

KARAT, C.-M. & KARAT, J. (1992). *Some Dialogue on Scenarios (Introduction to Special Issue on "Scenarios in HCI").* ACM SIGCHI Bulletin, 24 (4), 7.

KIERAS, D. (1988). *Towards a Practical GOMS Model Methodology for User Interface Design.* In M. Helander, Ed. *Handbook of Human-Computer Interaction*, 135-157. Elsevier Science Publishers BV: North Holland.

KIERAS, D. & POLSON, P. (1985). *An Approach to the Formal Analysis of User Complexity.* International Journal of Man-Machine Studies, 22, 365-394.

LEWIS, C., POLSON, P., WHARTON, C. & RIEMAN, J. (1990). *Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces.* In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, 235-242.

MACLEAN, A., YOUNG, R.M., BELLOTTI, V. & MORAN, T. (1991). *Questions, Options, and Criteria: Elements of Design Space Analysis*. Human-Computer Interaction, 6 (3 & 4), 201-250. Special Issue on Design Rationale, (Eds.) Carroll J.M. and Moran T.P.

PEDERSEN, E.R. & SUCHMAN, L. (1991). *Formalization in CSCW*. Panel at the Second European Conference on Computer-Supported Cooperative Work, Amsterdam, 113. Kluwer.

POLSON, P. & LEWIS, C. (1990). *Theory-Based Design for Easily Learned Interfaces.* Human-Computer Interaction, 5, 191-220.

POLSON, P.G., LEWIS, C., RIEMAN, J. & WHARTON, C. (1992). *Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces*. International Journal of Man-Machine Studies, 36 (5), 741-773.

RIEMAN, J., DAVIES, S., HAIR, D.C., ESEMPLARE, M., POLSON, P. & LEWIS, C. (1991). *An Automated Cognitive Walkthrough*. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, 427-428.

ROWLEY, D.E. & RHOADES, D.G. (1992). *The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure*. Proc. CHI'92: Human Factors in Computing Systems, Monterey, CA, 389-395, ACM Press: New York.

SHIPMAN, F.M. & MARSHALL, C.C. (1994). *Formality Considered Harmful: Experiences, Emerging Themes, and Directions*. Xerox Palo Alto Research Center, Technical Report ISTL-CSA-94-09

SHUM, S. (1991). *Cognitive Dimensions of Design Rationale*. In D. Diaper and N. V. Hammond, Ed. *People and Computers VI: Proceedings of HCI'91*, 331-344. Cambridge University Press: Cambridge.

SIDDIQUI, J.I.A. (1985). *A Model of Program Designer Behaviour*. People and Computers: Designing the User Interface, CUP: Cambridge.

SUBRAHMANIAN, E., KONDA, S.L., LEVY, S.N., REICH, Y., WESTERBERG, A.W. & MONARCH, I. (1993). *Equations Aren't Enough: Informal Modelling in Design.* Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 7 (4), 257-274.

VISSER, W. (1990). *More or Less Following a Plan During Design: Opportunistic Deviations in Specification*. International Journal of Man-Machine Studies, 33 (3), 247-278.

WHARTON, C., RIEMAN, J., LEWIS, C. & POLSON, P. (1994). *The Cognitive Walkthrough Method: A Practitioner's Guide*. In J. Nielson and R. Mack, Ed. *Usability Inspection Methods*, 105-140. Wiley: New York.

WINOGRAD, T. AND FLORES, F. (1986). *Understanding Computers and Cognition*, Reading MA: Addison-Wesley.

YOUNG, R.M. & BARNARD, P. J. (1987) The use of scenarios in Human-Computer Interaction: Turbocharging the tortoise of cumulative science. In J. M. Carroll & P. P. TANNER, Eds. *Proceedings of CHI+GI'87*, 291-296, ACM: New York.

YOUNG, R.M., GREEN, T.R.G. & SIMON, T. (1989) Programmable user models for predictive evaluation of interface designs. In K. BICE & C. LEWIS, Eds. *Proceedings of CHI'89*, 15-19, ACM: New York.

Appendix: IL description of Windows example

OBJECTS

```
window: window1, window2.
size:   full, half, three-quarters, one-quarter.
location: at-top, at-three-quarters, at-half, at-quarter, at-bottom.
screen-object:  resize-box1, resize-box2, ....
```

We have chosen a representation that only considers vertical size, and ignores width.

FUNCTIONS

```
size-of(window) = size.                    }
loc-of-top(window) = location.             } detected by looking
resize-box-of(window) = screen-object.     }
```

PREDICATES

```
on-screen(screen-object)                   -- detected-by-looking
```

OPERATIONS

```
operation move-window (window:W, location:Newloc)
  user-purpose:        loc-of-top(W) = Newloc
  device-command:move-window-command(W, loc-of-top(W) = Newloc)

operation expose-resize-box (window:W)
  user-purpose:  on-screen(resize-box-of(W))
  device-command:move-window-command(W,on-screen(resize-box-of(W)))

operation resize (window:W, size:Newsize)
  user-purpose:        size-of(W) = Newsize
  precond:             on-screen(resize-box-of(W))
                       & will-be-on-screen(resize-box-of(W))
  device-command:      resize-command(W, Newsize)
```

There is no clear way to state the requirements that the sizing box must be on screen both before and afterwards. The objectively correct condition is: loc-of-top(W) > Newsize.
The difficulty in expressing this condition is an indication of the problem users might have estimating where the re-size box will be at the end.

USER KNOWS INITIALLY

We do not need to specify anything here, as everything is on the display.

TASK

First

```
1.  size-of(window2)    = three-quarters.
```

then

```
2.  size-of(window2)    = half.
    loc-of-top(window2) = at-half.
```

There are two tasks to be performed in sequence: first making the bottom window bigger, then making it smaller again

DEVICE COMMANDS

```
resize-command (window:W, size:Newsize).
  The user can drag the resize box of a window until the window is in some desired
  size.  The resize box has to be on the screen throughout this process.

move-window-command (window:W, criterion:C).
  The user can drag the title-bar of a window until the window is in a position such
  that some criterion is met.  The criterion C has to be evaluable by visual means.
```

Note. The resize box will be on the screen provided that: size-of(W) < loc-of-top(W), not allowing for any overlaps of windows.

INITIAL DEVICE STATE

```
size-of(window1)    = half.
loc-of-top(window1) = at-top.

size-of(window2)    = half.
loc-of-top(window2) = at-half.
```

DISPLAYED

Various parts of a window, such as the resize-box, are displayed on the screen, or hidden off-screen, as the window is moved around.
