View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UCL Discovery

Preprint. Final version available as:

BLANDFORD, A. & GREEN, T. R. G. (2008) Methodological Development. In Cairns, P.A., & Cox, A.L. (eds.) *Research Methods for Human Computer Interaction*. CUP. 158-174.

Ann Blandford and Thomas Green

Methodological Development

Overview

One feature of Human–Computer Interaction research is that a broad range of methods have been developed to support the design and evaluation of interactive systems. In this chapter, we focus on evaluation methods – not because they are more important than design methods, but because we have more experience of them. In particular we consider the excitements and challenges of research on developing and testing new methods for evaluating interactive systems.

Evaluation methods can be broadly classed on three dimensions:

- With or without the active involvement of users;
- With or without a running system;
- With or without a realistic context of use.

Most methods fit into one position in this three-dimensional space. For example, controlled experiments typically involve the active participation of users with a running system in a laboratory setting (a non-realistic context of use), while Contextual Inquiry (Beyer and Holtzblatt, 1998) involves users with a running system in a realistic context. Conversely, interviews involve users but typically not a running system (though they may sometimes take place within the intended context of use). Most of the techniques that involve the active participation of users have drawn on and adapted methods initially developed within the social sciences, and methodological development has been relatively minor and definitely incremental.

In this chapter, we focus on the other set of methods – those which do not require the active involvement of users, being applied, rather, by HCI experts. These experts are not necessarily members of the original design team: they may be second-generation designers looking for a way forward; alternatively, they may be informing purchasers choosing between competing offerings.

These methods are usually referred to as either "analytical" or "inspection". The earliest analytical methods, such as GOMS (Card, Moran and Newell, 1983) and Task Action Grammar (TAG: Payne and Green, 1986) can trace their roots back to research in cognitive science. Although widely regarded as methods, the early emphasis in these approaches was on the notation and on describing a system in terms of that notation, then reasoning about it using the underlying theory on which the notation was based. Let us take these two examples a little further.

GOMS is based on a theory of human problem solving, focusing on how experts structure their goals and make choices between alternative courses of action. Average timings for basic actions, both physical and mental, have been established from empirical data; these can be used to calculate typical times for expert users to achieve specified goals. These calculations can, in turn, be used to compare alternative interface designs (e.g. Gray, John and Atwood, 1993), or to identify points where a system's efficiency is low. John and Kieras (1996) outline methods for performing GOMS analyses and discuss in detail the range of insights a GOMS analysis can yield about a system. Kieras (1997) goes further in terms of articulating a clear method for applying one particular variant of GOMS (NGOMSL).

TAG also considers users' tasks, but is based on a theory of how learners generalise from what they have already learnt, to make reasonable guesses at what they don't know. In a highly systematic or 'consistent' interface language, guessing is easy, because tasks with similar semantics have similar commands (e.g. 'move pointer 1 *place* forward' => CMD-F, 'move pointer 1 *word* forward' => CMD-SHIFT-F) ; in a very unsystematic, arbitrary one, guessing is impossible. TAG used a phrase-structure grammar to describe the interface language, then a simple semantic description of tasks and a two-level attribute grammar to summarise the first grammar: a consistent interface language could be summarised by a very short two-level grammar. Evaluating a design therefore required the users' tasks to be summarised in a semantic dictionary and the interface language to be restated as a two-level grammar entered via that dictionary, a hefty process resulting in a very detailed description but not easily compared to any other such evaluation, nor yielding much insight as to how the target design could be improved. Schiele and Green (1990) achieved a fairly thorough analysis of a real-life system, but the approach has been little used subsequently.

Cognitive science has been the theoretical foundation for many of the analytical methods developed to support reasoning about Human–Computer Interaction. As well as GOMS and TAG, as outlined above, cognitively oriented techniques include Task Knowledge Structures (TKS: Johnson, 1992), Cognitive Walkthrough (Wharton et al 1994) and CASSM (Blandford et al 2005). Other approaches, such as Thimbleby's (2004) work on matrix algebra and methods based on formal notations of computer systems (see Harrison chapter), draw more directly on theories of computing rather than cognition.

Another body of methods that do not require the direct involvement of users is checklist-based approaches (e.g. Heuristic Evaluation, Nielsen, 1994). These typically draw less directly on any theory, being based largely on the craft skill and experience of practitioners. Such methods involve inspection but minimal analysis.

In this chapter, we consider the issues involved in developing new analytical methods for evaluation in HCI. This is surprisingly poorly articulated in the literature on methods. Sometimes, such methods and notations appear not to have been designed or developed systematically, but to have happened more by accident. Maybe this is natural: method development is itself a form of design. Cross (2004) presents a compelling argument that design is nothing so simplistic as 'top-down' or 'bottom-up', as in the text book descriptions of authors such as Pahl and Beitz (1984), but instead is a much more complex process of choosing where to invest effort, moment by moment – a process that may seem haphazard to the onlooker but is in fact quite rational. Top-down design can be viewed as goal-driven: systematically working from the current situation to the goal state; conversely, bottom-up design is driven by the resources and ideas that are currently available.

The paucity of accounts of how methods have been developed in the past represents a challenge in terms of presenting a method for developing methodologies. Unlike most other chapters of this book, there is minimal literature to base the method description on. There are, however, three sources of evidence on which we can draw to develop our account of method development:

- one is the snapshots of methods as they are presented in the literature over time – for example, the different descriptions of Cognitive Walkthrough as it evolved from 1990 (e.g. Lewis *et al*, 1990) to the present (e.g. Blackmon *et al*, 2003; Allendoerfer *et al*, 2005);
- a second is to draw on the literature on design. Method development is, as noted above, a form of design.
- finally, our most compelling source of evidence is personal reflection: we have been directly involved in the development of at least ten methods and notations between us, and have therefore built up substantial experience of method development and testing:
    - TAG (Payne and Green, 1986), as described above;

- Cognitive Dimensions of Notations (CDs: Green, 1989; Green and Petre, 1996; Green *et al*, 2006), a language for discussing properties of notations (particularly programming languages);
- Programmable User Modelling (PUM: Young, Green and Simon, 1989; Blandford and Young, 1996), an approach based on describing the knowledge a user would need to use a device, based on a model of human problem solving;
- Entity–Relationship Modelling of Information Artefacts (ERMIA: Green and Benyon, 1996), a more formal approach to describing information artefacts;
- Interacting Cognitive Subsystems (ICS: May, Barnard and Blandford, 1993; Barnard and May, 1999), a more abstract approach to reasoning about a user's cognition while working with a system;
- Interaction Framework (IF: Blandford, Harrison and Barnard, 1995), an approach that aims to abstract away from details of the individual actors (people and computer systems) and focus on the patterns of interaction between those actors;
- Evaluating Multi-modal Usability (EMU: Hyde, 2002), a method that focuses particularly on the ways multiple modalities are recruited in an interaction;
- Distributed Cognition for Teamworking (DiCoT: Furniss, 2004; Furniss and Blandford, 2006), a methodology based on the ideas of Distributed Cognition (Hutchins, 1995; Hollan, Hutchins and Kirsh, 2000);
- CASSM (Blandford *et al*, 2005; Connell *et al*, 2004), as described below in the case study; and
- Claims Analysis for information systems (Blandford, Keith and Fields, 2006), an adaptation of Claims Analysis (Carroll and Rosson, 1992) that tailored it towards the design of digital libraries and similar information systems.

As this list makes apparent, our experience is entirely in developing evaluation methods rather than (for example) design methods, and this bias permeates the rest of this chapter.

In presenting the method below, we draw on these different sources of evidence to construct what we hope is a coherent account of the principles underpinning methodology development. You should note that we tend to use the terms 'method' and 'methodology' interchangeably. We tried sticking with one of these terms rigorously, and found we could not – for example, some authors refer to Cognitive Walkthrough as a method and others as a methodology, and when we adopted their terminology we inherited inconsistencies. Broadly, we view a methodology as a collection of methods used in a systematic way, but of course at a higher level of abstraction that collection of methods can be regarded as a 'bigger' method.

The Method

Developing and testing a method is not quick. Neither is it for the faint-hearted. As is indicated by the spread of dates on the citations for some of the methods outlined above (notably GOMS, Cognitive Walkthrough, Cognitive Dimensions), method development has often taken place over many years. Nevertheless, substantial work can be done within the limits of a Masters or Doctoral project. For example, DiCoT (Furniss, 2004) was developed as a Masters project, using ambulance control as a case study, though it has subsequently undergone further testing in a different domain – namely agile software development (Sharp *et al*, 2006). Similarly, EMU (Hyde, 2002) was developed and tested as a doctoral research project. It is also possible to test a methodology developed elsewhere, as John and Packer (1995) did with Cognitive Walkthrough, and refine or extend it, as we did with Claims Analysis (Blandford *et al*, 2006), so that any one project does not necessarily have to cover all phases of the development lifecycle.

In simple terms, the development of a method involves some key phases of activity: while we present them here in a linear fashion, the actual development of a methodology will typically be

iterative (with testing leading to the identification of further requirements), and phases may sometimes be interleaved in apparently arbitrary and chaotic ways. The core phases of development are:

- Identification of an opportunity or need;
- Development of more detailed requirements (optionally);
- Matching opportunities, needs and requirements;
- Development of the method; and
- Testing of the method.

We consider each of these phases of activity in more detail here, and present a single example of the development of a methodology (CASSM) in the next section.

Identification of an opportunity or need

As with any design, the development of a methodology starts with either an opportunity or a need. For example, a cognitive or computational theory (e.g. the matrix algebra underpinning the work of Thimbleby (2004) or the Distributed Cognition theory that formed the basis of DiCoT (Furniss and Blandford, 2006)) might suggest a new way of analysing a system to yield new and valuable insights. Conversely, there might be a need to evaluate a system that is not supported by current methods: our development of Claims Analysis (Blandford *et al*, 2006) fits into this category. In that development, we were seeking an approach that: would support digital libraries developers in thinking about their design from a user's perspective; allowed us to codify an understanding of cognition and information seeking; and fitted within their ongoing development practice.

In an ideal world, opportunity and need will come together. This was the case for CASSM (as described below), in that we drew on existing resources (our earlier work on CDs, ERMIA and PUM), but also had reasonably clear goals for what we wanted to achieve with our new approach.

Development of more detailed requirements

Developing a new methodology is expensive, in time and resources. It is important to consider what future benefits will accrue from that investment. That sounds a mercenary way to think about research, but a method has to have some benefits. Those may be direct benefits, in terms of a method being taken up and used in practice and hence influencing the designs of future systems, or they may be indirect, e.g. helping researchers to better understand the challenges inherent in developing methodologies, or being a vehicle for encapsulating a theory of cognition. Whatever the goals of a methodology development project, it is a good idea to be reasonably clear about what they are. If you are really, sincerely, trying to develop an approach that is to be taken up and used in practice, then it is important to understand the requirements of practice. If, on the other hand, your interests are more theoretical (e.g. to explore the feasibility of applying entity–relationship modelling to the design or evaluation of information artefacts (Green and Benyon, 1996)) then it may be less important to identify more detailed requirements on the method.

There is still no clear set of requirements for HCI methodologies to be taken up in practice. Some of the requirements that have been discussed are as follows. Some of these apply equally to both design and evaluation methods; others apply particularly to evaluation methods.

Validity: For analytical evaluation methods, it is important that the method should, as far as possible, support the analyst in correctly predicting user behaviour or identifying problems that users will have with the system, minimising the number of false positives (issues that are identified as problems that actually are not) and misses (failures to identify actual problems). Of course, some issues will be outside the scope of a particular evaluation method (e.g. a matrix algebra approach is unlikely to have much to say about the effectiveness of team working), so another requirement concerns scope.

Scope: The scope of a method, in terms of the kinds of issues it does and does not address, should be clear to users of the method. This is important so that analysts can both select an appropriate method for addressing their current concerns and also appreciate the limitations as well as the value of their analysis.

Reliability: The extent to which different analysts applying the same method will achieve the same results (the same design or evaluation insights) is considered by some to be an important consideration (e.g. Hertzum and Jacobsen, 2001).

Productivity: In the past, simple problem count (the number and/or severity of problems an evaluation method helped identify) was considered an important criterion. Since the work of Gray and Salzman (1998), this criterion has been relatively de-emphasised. Nevertheless, any method should somehow either spur developers, or else assist users, trying to choose whether to adopt a particular device.

Usability: Surprisingly little discussion has focused on the idea that HCI methods must themselves be usable. This means that they must fit within design practice (one might want to be more specific about what particular kinds of design practice are to be addressed; for example, the practices of safety critical systems development are normally different from those of website development).

Learnability: How easy it is for a practitioner to pick up and work with a method will be a strong determinant of take-up and use in practice. We are not aware of many studies of the learnability of methods in HCI, although Blandford, Buckingham Shum and Young (1998) studied the practicalities of teaching (and learning) PUM, and John and Packer (1995) investigated the learnability of Cognitive Walkthrough.

Insights derived: Wixon (2003) argues strongly that the most important criterion for any HCI method (he focuses on evaluation methods, but the same argument surely applies to design methods) is whether or not it gives the designer insights that will help to improve design. Never mind how valid, reliable or productive a method is: does applying it result in a better system?

Wixon's (2003) work, in particular, highlights the fact that there is still little agreement on what features an HCI method should possess. The list of possible requirements presented here is by no means complete; some of them may, indeed, be regarded as unimportant or irrelevant to future HCI practice. Also, as noted above, for many methodology development projects, practitioner-oriented requirements may be of secondary concern to scientific requirements such as the feasibility of encapsulating a particular theory in a method at all.

Matching opportunities, needs and requirements

Whatever the starting point (need or opportunity) and requirements, it is then necessary to consider what theories, tools or resources are available for addressing the need (or, conversely, to identify what unrecognised needs a new opportunity addresses). This is typically an exploratory step.

If you start with a need, then the question is: how might that need be addressed? If we dismiss as unrealistic the possibility that the answer will emerge as if by magic, this phase typically involves searching through work that has been done before to find likely candidate approaches, then testing them to establish how well they address the need. For example, when we were looking for suitable methods for supporting digital library development, we started by testing some widely used methods, namely Cognitive Walkthrough and Heuristic Evaluation. This involved applying these methods ourselves to selected digital libraries and also running some user studies to find out what difficulties the users of those libraries were actually experiencing, to establish how well the methods were supporting reasoning about real user difficulties. We found that the methods were

delivering useful insights about surface difficulties (e.g. poor labelling, or unhelpful help messages), but not giving leverage on why real users were having such difficulty working effectively with the digital libraries (Blandford *et al*, 2004). We therefore scoured the literature to find less familiar methods that might fit our needs better. Claims Analysis (Carroll and Rosson, 1992) seemed to fit the bill in that it was intended to be used within an ongoing design process (good!) and focused on the positive and negative effects of design decisions on the user, which looked like a promising approach. Blandford *et al* (2006) report on our experience of learning, applying and adapting Claims Analysis; the key points to emerge from that study were that: those learning a new technique are highly dependent on published descriptions of the approach, unless they have direct access to the method developers; and that there were substantial cultural gulfs between ourselves (as human factors specialists) and the digital library developers with whom we worked, in terms of prior knowledge, expectations and values. These differences had to be accommodated in the way that we developed and communicated the approach with them. In summary, then, the identification of possible approaches for addressing a need is typically highly exploratory, finding and testing work (often by others) on which to build.

If you start with an opportunity, or resource, the exploration is typically less concerned with finding appropriate theory and more with finding ways to develop theory into a method. This may involve minimal exploration, moving swiftly on to the next step of developing the method. However, it is often helpful to have identified existing methods that can be adapted to encapsulate the new theory. For example, having synthesised the elements of Distributed Cognition from multiple sources, Furniss (2004) searched for data gathering and representation methods to support the approach of reasoning about a work context in terms of the constructs of Distributed Cognition. After extensive reading, he adapted the method and representations of Contextual Design (Beyer and Holtzblatt, 1998) to develop DiCoT; these were chosen because they matched his requirements well.

This phase of matching needs, requirements and opportunities can be regarded as a process of gathering together suitable ingredients for developing the method. Whether particular ingredients fit well together is often a matter of trial and error, explored within this phase or within development or testing of the method.

Development of the method

The development of a method is the creative step that is the hallmark of all design. As such, there is relatively little that can be written about it in terms of structured processes (though drawing on examples of existing methods that share similar features is often a good place to start). Development is itself often iterative: for example, investigating the strengths and weaknesses of particular gathering data techniques (about users or computer systems or work contexts or some combination of these elements), or the effects of changing the representations used or the detailed analysis techniques, or the tutorial or other description of how to apply the method. In the account of how we developed CASSM (below) we recount some of the main design decisions we made, and changes we subsequently made, but we are unable to clearly articulate where each of the ideas that have gone into the method came from – only how we implemented and tested those ideas. We describe some of the insights we had in terms of 'eureka!' moments: we can't tell you what made those moments happen.

Testing of the method

As outlined above, there may be many kinds of requirements on the method that has been developed. What kinds of testing are needed will depend on what the requirements are. For some of the cognitively oriented techniques, such as GOMS, the primary concern, particularly in the earlier stages of development, was cognitive validity: that the findings from a GOMS analysis should match as closely as possible the findings from empirical studies of experts working with

the same systems. Through the 1980s and early 1990s, the emphasis for testing was on productivity (e.g. Desurvire *et al*. (1992), Sears (1997), Cuomo & Bowen (1994)) – an emphasis that was criticised on scientific (though not on practical) grounds by Gray and Salzman (1998). As outlined above, other criteria for testing have become more widely recognised (though not yet universally agreed) in recent years.

One way to think about testing a method is to consider the following six questions (which we will refer to as the "PRET A Rapporter" framework):

1. **P**urpose of evaluation: what are the goals of the testing, or the detailed questions to be answered in the study? These will reflect the needs and requirements that the method was developed to address.
2. **R**esources and Constraints: what resources are available for conducting the study and, conversely, what constraints must the study work within? For testing methods, this question usually presents the most challenges, so we consider it in more detail below.
3. **E**thics: what ethical considerations need to be addressed? In method testing, the main ethical concerns are likely to relate to privacy of participant information. We do not consider this in more detail here.
4. **T**echniques for gathering data must be identified. See other chapters of this book for a discussion of data gathering techniques.
5. **A**nalysis techniques must be selected. Similarly, analysis techniques are discussed elsewhere in this book.
6. **R**eporting of findings is (usually) the final step. Again, writing is discussed elsewhere in this book.

Resources and constraints present the greatest challenge: in particular, recruiting participants to test a method is important because how a method is used depends so heavily on who is using it. The knowledge and expertise of the analysts working with the method simply cannot be ignored. Hertzum and Jacobsen (2001) refer to this as the "evaluator effect". In early tests of methods (many years ago), people tended to ignore the evaluator effect – indeed, this was one of the criticisms made by Gray and Salzman (1998) of those early studies. Nielsen and Landauer (1993) turned this issue up-side-down and asked how many evaluators you needed to be reasonably sure you had identified most of the important usability problems with a system (the widely quoted answer is 5, though the origins of that number are disputed and, as Woolrych and Cockton (2001) argue, that answer makes many presumptions about the spread of evaluator expertise; it also assumes the value of productivity as a way of assessing the value of a method).

Many of the tests of methods that have been conducted are comparative: they have pitted methods against each other. This is a fine thing to do (we have done it ourselves), provided that you accept that such a comparison can never be scientifically 'clean': if different analysts use each method then the individual differences of the analysts is a confounding variable; if the same analyst applies the different methods then as they work they will gradually learn about the systems being evaluated, and they may have different levels of skill with the different methods – it is simply not possible to conduct scientifically rigorous comparison of the style discussed in Chapter N of this book. However, qualitative studies (e.g. Connell *et al*, 2004) that compare the experiences and outcomes of applying different methods can yield valuable insights into their strengths and limitations.

You may only want to test an individual method, in which case probably the dominant concern is with ecological validity. How much does it matter that the method should be used by practitioners, or will (for example) HCI students serve as a good surrogate? If the concern is with learnability

(e.g. John and Packer, 1995; Blandford *et al*, 1998) then working with students seems a reasonable approach, and the next question is whether to work with a very small number or with a larger group (studies have been done with a single student, tracking their learning and use of resources in detail). Recruitment and scheduling of training sessions are likely to be the next issues to consider. How you address these will very much depend on circumstances: on the timing of the study and the available resources.

If the testing really needs practitioner involvement then what is possible depends on the duration and level of commitment required of those practitioners. It is important to consider the costs and benefits to the practitioners of involvement in any study. In our experience, few practitioners have much time to commit to learning or applying a new method unless it fits snugly with their existing practices and gives clear benefits. Certainly, longitudinal studies (e.g. Blandford *et al*, 2006) are demanding, on both investigators and practitioners, though in our experience they are also enjoyable and enlightening.

Summary

It is impossible to be prescriptive about the method for developing a method. There are huge variations in the motivations for developing a method and the contexts within which it happens – and hence the available resources and constraints under which that development and subsequent testing occur. In this section, we have aimed to highlight what are, in our experience (and also based on our readings of other people's work in this area), the important phases of activity and key issues to consider in developing a method. In the next section, we present an account of our development and testing of CASSM to date, as an example of method development.

Applying the Method

Concept-based Analysis of Surface and Structural Misfits, or CASSM, is an approach to assessing the 'quality of fit' between the way users think about their activity and the way a system represents it. We have been developing the approach over about eleven years at the time of writing, although most of that time progress has been slow because the development has been in our spare time. CASSM started life as UUUM (Usable, Useful, Used Modelling – an ambitious, but rather non-specific name), then became OSM (Ontological Sketch Modelling – a name that we explain below) before becoming CASSM (pronounced "chasm")– a name that is intended to evoke the 'gulf' between user and system.

The development of CASSM has been, in Suchman's (1987) terms, "situated". Like her canoeist shooting the rapids, we knew where we were aiming, we had some initial ideas of how to proceed, but each particular development manoeuvre was constructed in response to the current situation: to what we had found out so far through empirical studies and personal reflections and to the opportunities that presented themselves. The requirements for CASSM were hatched somewhere unlikely – we now disagree about whether it was in a pub in Sheffield or on a bus in Norway. Either way, we agreed that there was a need for an evaluation method that…

- Would be useful: it should reveal important things about usability of tools such as drawing packages that were not well suited to task-based analyses: what makes a drawing package hard to use is typically not the task structures but the appropriateness and obviousness of the concepts – such as drawing objects, layers, 'handles' and fills – with which the user has to work to achieve what they want.
- Would be usable, treading an appropriate path between the 'death by detail' that characterises many rigorous evaluation approaches (including the ERMIA and PUM on which we were working at the time) and the vagueness of CDs and checklists.
- Would ultimately be used by people other than its developers and their close friends.

'Useful, usable, used': we conclude this section with our current assessment of how well we have met those objectives.

With our backgrounds in modelling (and methods based on the construction of models), it was perhaps inevitable that we would think of our method as involving some modelling work, although we did not want that modelling to be onerous: we had learnt through our experiences of trying to teach ERMIA and PUM that sophisticated modelling activity is not an attractive proposition to many people, and that the initial learning curve for any new method would have to be relatively small. The fact that we wanted to focus on concepts and relationships between them (to provide a complement to the many task-oriented methods that were available at the time) led us to the second name for the approach: that we were interested in the *ontology* with which users were working; that modelling was *sketchy*; and that it was *model-based* – hence, Ontological Sketch Modelling (OSM).

The first version of OSM focused on concepts and relationships. Examples of concepts and relationships are that layers and drawing-objects are concepts, and they are related by a drawing-object being on a layer. We were interested in reasoning about Green's CDs, many of which are concerned with how easy it is to make changes to systems, and initially we tried to do this just by focusing on concepts and relationships; after weeks of struggling and arguing (the path of research collaboration does not always run smoothly), we eventually accepted that we would need some representation of actions (i.e. what actions need to be performed in order to create or delete entities, or to change attributes or relationships).

Drawing particularly on our experience with PUM, which included a complex representation of actions, our first attempt at representing actions in OSM involved describing them in detail – an approach that we abandoned reasonably quickly as being too time-consuming and having limited return on investment.

Early OSM analyses (including the first versions of it we taught to students) were entirely paper-based. This had the advantages of paper such as being flexible and easy to present in a classroom setting. However, we believed that the ideas would be easier to communicate (e.g. via the internet) and more likely to be taken up by others if we had tool support for the approach. Our first attempt at tool support, OSMosis, is shown in Figure 1. The process of designing OSMosis was immensely useful: it forced us to make explicit what we meant by terms (in a way that could be fudged on paper) and it forced us to think about what we really wanted to be able to say about different elements of the model. For example: what did we want to be able to say about an action? At the time of OSMosis, what we chose to say was what the action's name was, and whether or not it was moded or disguised: i.e. whether the same physical action would have the same effect on the system state under all conditions, and whether the user could easily discover and predict the effect of the action, because the examples we had been working on up to that point led us to believe that these were the important features of an action for reasoning about usability.

We could go on at length about the design of OSMosis, but will resist. The details really do not matter for the purposes of this chapter. The important points that we should note are that:

- We aimed to minimise premature commitment by allowing the user to create description 'stubs', with an explicit facility to mark when item descriptions were complete, and also to maximise flexibility of expression by creating a 'notes' field for every item.
- At this point in the development of OSM, we were explicitly naming all important actions, as well as entities, attributes and relationships.
- Although we had thought carefully about the interface design for OSMosis (as well as about what information the user should be able to enter into the system), when we started using it 'in earnest' (such as for the system description shown in Figure 1) we found it tedious to work with: too many windows, and poorly represented connections between concepts (e.g.

the idea that an attribute 'belonged to' a particular entity). We did not need to subject OSMosis to user testing with any independent users to recognise its limitations.

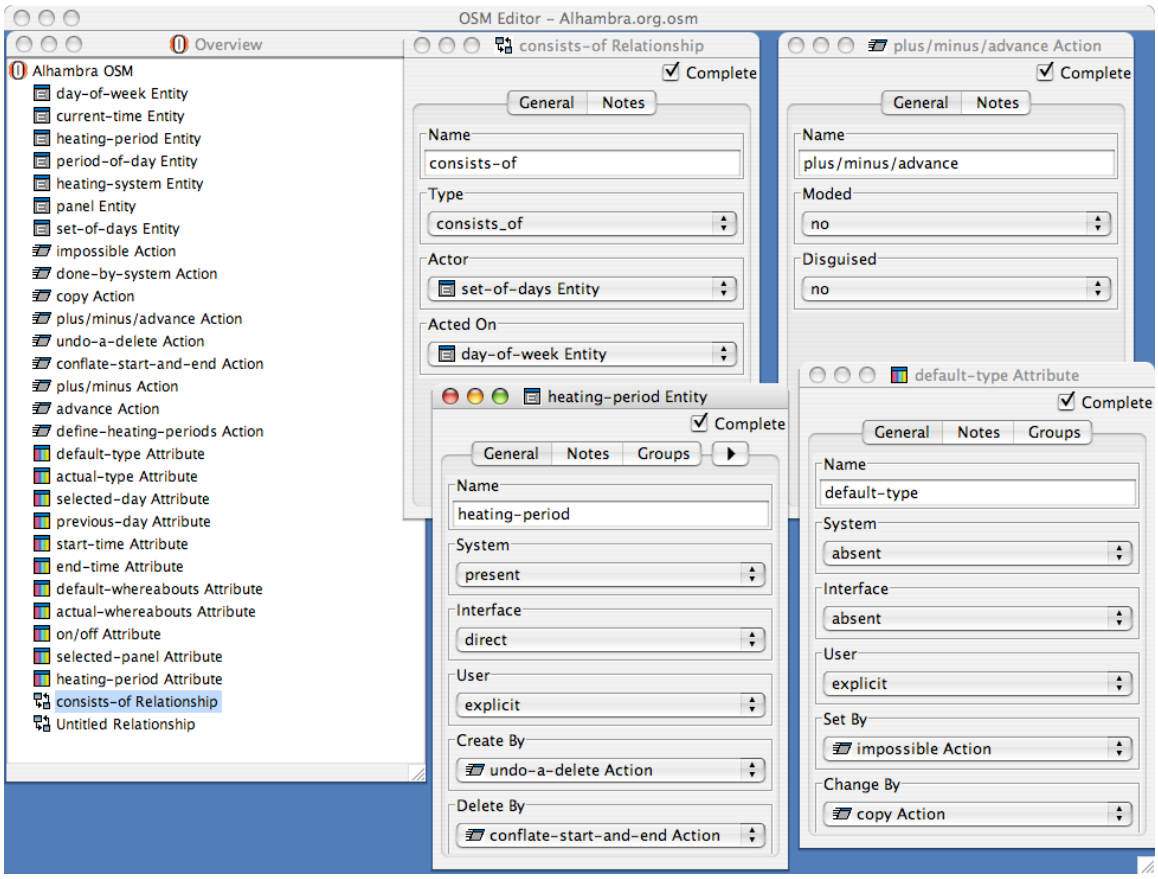

Figure 1: A screen shot of the OSMosis editor, showing example windows for entities, attributes, actions and relationships

As noted above, one of our interests was in supporting reasoning about some of Green's CDs. 'Repetition viscosity' is an example of a CD, describing 'resistance to small changes': for example, in most drawing applications, if you move a box in a boxes-and-connectors diagram then you have to separately move each one of the connectors to make them reconnect with the box. Our hope was to generate formal descriptions of those CDs in terms of the ontology of OSM – i.e. entities, attributes, relationships and actions. Somewhere in this process, we had a 'eureka' moment, where we realised that the representation was wrong: that we had to explicitly link attributes with their entities, and that the names of actions were unimportant, but that what mattered was how easy or otherwise it was to create or delete entities, or change the values of attributes. Actions *per se* disappeared from the representation, although their properties remained.

Another 'eureka' moment came when we realised that there were two broad classes of misfits that we could reason about with this representation. One was what came to be called 'surface' misfits, which were simple instances of concepts that mattered to the user that were not well represented in the system or, conversely, concepts that were central to the operation of the computer system that were difficult for the user to grasp. The other was 'structural' misfits: ones which related to the internal structure of the computer system representation which made activities that were conceptually simple to the user (like moving a box on a boxes-and-connectors diagram) difficult to achieve in practice (e.g. because of repetition viscosity). The surface misfits could be easily spotted just by 'eyeballing' an appropriately designed system description; the structural misfits

required some additional (computable) analysis. With OSMosis, this computable analysis was only achieved by exporting the description to a file which was subsequently analysed as a separate program. The 'eyeballing' was not supported at all.

By this point, another difficulty we were experiencing was with the name of the approach: the word 'ontological' was either unfamiliar to people or was associated with a particular approach to knowledge representation in artificial intelligence; the word 'sketch' made some people think we were only interested in interfaces to sketching (i.e. drawing) systems; and the word 'modelling' evoked links to the Unified Modelling Language (UML). All wrong! After much head-scratching and playing around with terms and acronyms, we opted to rename the approach as CASSM. We do not think this is the perfect name, but we have not yet conjured up a better one. At least this name is pronounceable ("chasm") and has not been found to evoke inappropriate associations.

These many insights were brought together in a refinement to the method and the implementation of a support tool that we called Cassata (see Figure 2). Cassata allows the analyst to describe the properties of actions (e.g. 'fixed', 'easy', 'indirect') but not name them, and includes a facility to automatically analyse a system description for various CDs; it also retains the idea of the analyst writing freeform notes to capture related insights or explanations, explicitly links attributes to the corresponding entities, and supports 'eyeballing' of the description to spot surface misfits (they are red (or dark if viewed in greyscale) in Figure 2). The complete package of a CASSM tutorial, the Cassata tool and several worked examples of analyses are available via the internet (www.uclic.ucl.ac.uk/annb/CASSM/). This was important to us: for any method to be taken up and used, it has to be accessible to potential users; the tutorial package includes all the materials a novice needs to start working with CASSM.

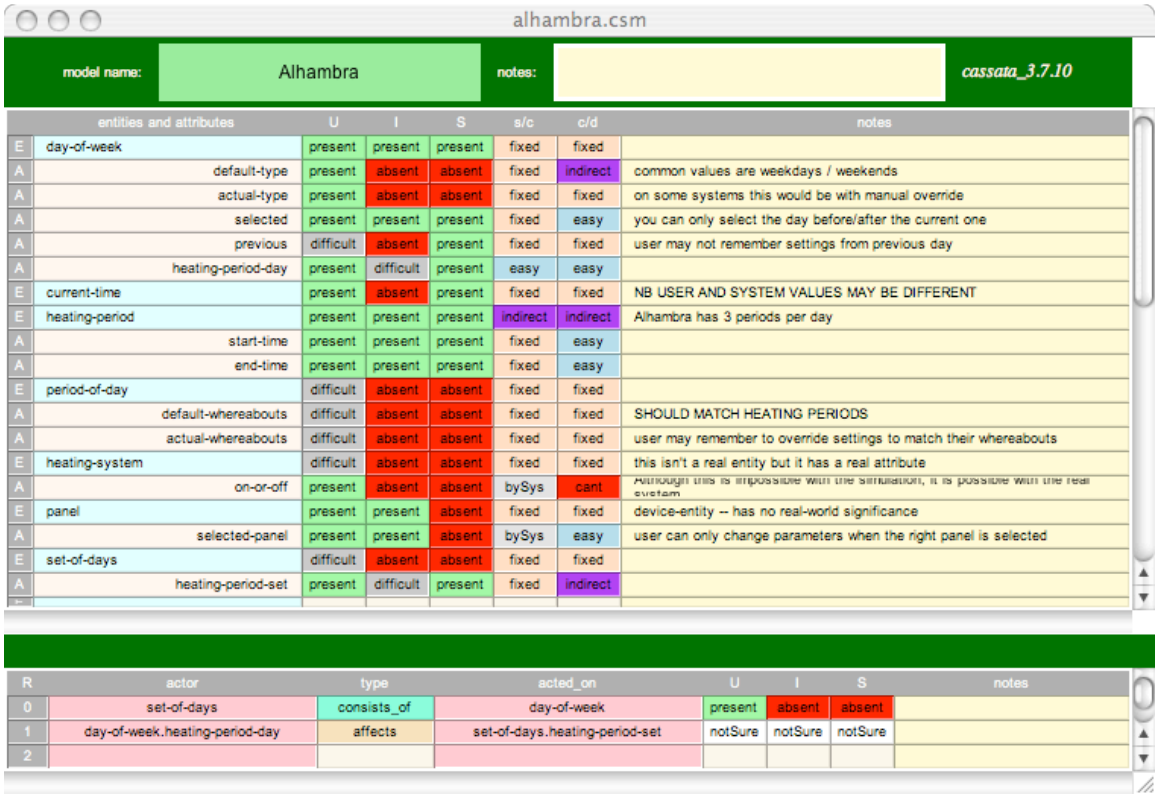

Figure 2: screen shot of Cassata, showing a complete system description

Throughout the development process, various forms of testing have been conducted. Many of these have been informal, largely concerned with the learnability of the approach – by teaching it

to students and presenting it as a conference tutorial and informally establishing what difficulties people have with learning and applying the approach. This kind of informal testing has been conducted since the earliest stages of development. In addition, analytical evaluations have been conducted to compare CASSM with evaluation approaches such as Cognitive Walkthrough (Connell *et al*, 2004) and others (as yet unpublished). In these evaluations, we have not been pitting approaches against each other to establish which is 'better', but to find out more about the scope of each (what kinds of systems and users each approach is suited to, and what kinds of usability issues it highlights). We have also engaged in testing the approach with software developers, and on various kinds of systems; this work is ongoing.

Overall, CASSM has undergone iterative development and testing, and we do not consider that process to be complete yet. In terms of the phases of activity discussed in the previous section:

- *Identification of an opportunity or need* started from the desire to develop a method that was useful, usable and used, and that gave the kinds of insights we have outlined about misfits, including CDs.
- *Development of more detailed requirements* has largely refined the initial need.
- *Matching opportunities, needs and requirements* involved drawing on our earlier work on ERMIA, CDs and PUM.
- *Development of the method* involved a lot of trial and error and eureka moments, as outlined above.
- *Testing of the method* has involved using it ourselves on a variety of systems, teaching it to others, and comparing the outputs of CASSM analysis with those of other methods.

All the development and testing have been conducted keeping the initial needs and motivations in mind. At the time of writing, we have established that CASSM is useful, we have a reasonable understanding of which elements of it are usable and which are (still) difficult to learn, and we have made some limited progress on it being used. The next focus will be on communicating the approach to practitioners and testing it more thoroughly for take up and use in practice.

Related Studies

To the best of our knowledge (and we have looked in all the obvious places and many non-obvious ones too), there are no accounts of how to develop methods, and few case studies of how methods have been developed. The only case study we are aware of is our own (Green *et al*, 2006) account of the development of Cognitive Dimensions. This dearth makes it difficult to present an account of related studies with any confidence. We can tell stories based on snapshots of methods as presented by their developers over the years, but cannot account for the detailed deliberations or serendipitous events that nudged the developments.

Green *et al* (2006) highlight the serendipity that has characterised the development of Cognitive Dimensions. From an initial aim of developing a way of describing relevant features of designs compactly and with limited ambiguity (Green, 1989), CDs came to be viewed as 'discourse tools' for information-based artefacts (Green, 1994) in which the dimensions were identified, described and refined largely from the personal experiences of the members of the development team. A publication on the application of CDs to visual programming languages (Green and Petre, 1996) unexpectedly led to them being regarded as only applying to that kind of system. Nevertheless, the vocabulary was sufficiently lightweight, flexible and expressive to be fairly rapidly taken up by others, particularly within the Psychology of Programming community. One of the effects of this – a point that we have not discussed previously in this chapter – has been that CDs have been adopted and adapted in ways that were not intended by their originator, resulting in offspring like CASSM, as described above, CiDa (Roast *et al*, 2000) and CD-oriented personas (Clarke, 2005). In the process, the set of dimensions has been debated and modified; while there remains a 'core'

to CDs as originally conceived, there is arguably no longer a single focus or direction to the development as CDs are taken up and used more widely.

Heuristic Evaluation (Nielsen and Molich, 1990) is another approach that has been taken up, adopted and adapted. In this case, it appears that the original set of ten heuristics were developed based on craft skill; subsequent sets of heuristics have been published for specific types of systems such as webs sites and e-commerce sites – again, apparently based on the experience of Nielsen and others (see www.useit.com). In the case of Heuristic Evaluation, the main evaluative focus has been on costs and benefits of the approach – a question addressed not just by Nielsen himself (e.g. Nielsen, 1995) but also by others (e.g. Woolrych and Cockton, 2000).

The adoption and adaptation that characterise the development of CDs and Heuristic Evaluation are less obvious in the development of other, arguably more complex, approaches. GOMS (Card, Moran and Newell, 1983) has been developed into four variants, each focusing on a slightly different aspect of user–system interaction, but all based on the same core representation and all assuming expert behaviour (John and Kieras, 1996). Various tools for supporting GOMS analysis have been developed by different research groups (Baumeister *et al*, 2000) to facilitate take-up and use within industry. Nevertheless, while GOMS has been applied to industrial software systems (e.g. Gong and Kieras, 1994), the published evaluations of GOMS focus on its validity (how well it can be used to predict actual expert behaviour) and not criteria such as learnability or fit with industrial practice, and we are not aware of any case studies of the use of GOMS by (non-academic) practitioners.

This contrasts with the development of Cognitive Walkthrough, where much less emphasis has been placed on validity and more on usability and utility of the method. The earliest papers on the approach are largely concerned with the theoretical foundations of the method, but through the early 1990s the developers were clearly working to achieve a balance between providing good support, encouraging rigour and avoiding inflicting mind-numbing boredom on analysts. They also focus directly on how to apply the method, rather than on constructing a model as such. Over the early years of development, the number and complexity of questions an analyst was expected to answer for every step of a walkthrough changed from nine relatively sophisticated questions (Lewis *et al*, 1990) to four more straightforward ones (Wharton *et al*, 1994), and more recently to three (Blackmon *et al*, 2002). Recent work has, like Heuristic Evaluation, focused on adapting the approach to apply particularly to website evaluation. Also, a case study on how the method needed to be simplified and adapted to fit with the constraints of commercial software development has been conducted and reported (Spencer, 2000).

In summary, different methodology development projects have had different focuses in terms of goals (the criteria against which they appear to evaluate their methods) and outcomes to date (e.g. extent and type of take-up and adaptation by others, whether other academics or commercial users). In all cases, it would appear that development has been situated, responding to opportunities that arise and exploiting the strengths and values of the individuals involved in the development projects. Some stages of development are explicit in the publications about various methods; others can only be inferred.

Critique

As should be obvious, the method for method development is under-defined. All attempts to define systematic design methods have constrained creativity and not proved that helpful for anything other than routine design. The design of new methodologies certainly is not routine. Nevertheless, good design typically contains certain ingredients (including, but not limited to, inspiration and insight). In this chapter, we have aimed to outline important activities, goals and techniques that contribute to methodology development.

Probably the biggest challenge is evaluating evaluation methods. As discussed above, there are many possible evaluation criteria including validity, scope, reliability, productivity, usability, learnability and insights derived from applying a method. In principle, any new method should be assessed against all relevant criteria; in practice, there has been limited work on assessing many methods against any of the criteria, never mind all of them. Indeed, there is scope for conducting informative and rewarding evaluations of methods as separate research projects, and there is a need for innovative methods for evaluating methods.

Stepping back to consider the whole development cycle again, we are not yet in a position to critique the approach we have outlined and exemplified. However, we hope that this chapter is informative and inspirational for others who are up to the challenge of developing or testing methods to address future needs in HCI.

References

Allendoerfer, K., Aluker, S., Panjwani, G., Proctor, J., Sturtz, D., Vukovic, M. and Chaomei Chen (2005) Adapting the cognitive walkthrough method to assess the usability of a knowledge domain visualization. *INFOVIS 2005. IEEE Symposium on Information Visualization*, 195-202, 23-25 Oct. 2005.

Barnard, P. J. and May, J. (1999) Representing Cognitive Activity in Complex Tasks. *Human–Computer Interaction Journal*. 14.1-2. p93-158.

Baumeister, L. K., John, B. E., and Byrne, M. D. (2000) A comparison of tools for building GOMS models. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York, NY, 502-509.

Beyer, H. & Holtzblatt, K. (1998) *Contextual Design.* San Francisco : Morgan Kaufmann.

Blackmon, M. H., Polson, P. G., Kitajima, M. & Lewis, C. (2002) Cognitive Walkthrough for the Web. *2002 ACM conference on human factors in computing systems (CHI'2002),* 463-470.

Blackmon, M. H., Kitajima, M., and Polson, P. G. (2003) Repairing usability problems identified by the cognitive walkthrough for the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing System*. ACM Press, New York, NY, 497-504.

Blandford, A. E., Buckingham Shum, S. & Young, R. M. (1998) Training software engineers in a novel usability evaluation technique. *International Journal of Human-Computer Studies*, 45.3. 245-279.

Blandford, A., Green, T. & Connell, I. (2005) Formalising an understanding of user–system misfits. In R. Bastide, P. Palanque & J. Roth (Eds.) *Proc. EHCI-DSVIS 2004*. Springer: LNCS 3425. 253-270.

Blandford, A. E., Harrison, M. D. & Barnard, P. J. (1995) Using Interaction Framework to guide the design of interactive systems. *International Journal of Human-Computer Studies*, 43. 101-130.

Blandford, A., Keith, S., Connell, I. & Edwards, H. (2004) Analytical usability evaluation for Digital Libraries: a case study. In *Proc. ACM/IEEE Joint Conference on Digital Libraries*. 27-36.

Blandford, A., Keith, S. & Fields, B. (2006) Claims Analysis 'in the wild': a case study on digital library development. *International Journal of Human–Computer Interaction.* 21.2. 197-218.

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

Card, S. K., Moran, T. P. & Newell, A. (1983) *The Psychology of Human Computer Interaction*, Hillsdale NJ: Lawrence Erlbaum.

Carroll, J. M. & Rosson, M. B. (1992) Getting around the task-artifact cycle: how to make claims and design by scenario. *ACM Transactions on Information Systems*, 10.2. 181-212.

Clarke, S. (2005) Describing and measuring API usability with the cognitive dimensions. *Proc. Cognitive Dimensions of Notations 10th Anniversary Workshop*. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/workshop2005/Clarke_position_paper.pdf (viewed 14/10/06).

Connell, I., Blandford, A. & Green, T. (2004) CASSM and cognitive walkthrough: usability issues with ticket vending machines. *Behaviour and Information Technology*. 23.5. 307-320.

Cross, N. (2004) Expertise in Design: an overview. *Design Studies*. 25. 427-441.

Cuomo, D.L. & Bowen, C.D. (1994) Understanding usability issues addressed by three user-system interface evaluation techniques. *Interacting with Computers*, 1994, 6.1. 86-108.

Desurvire, H.W., Kondziela, J.M. & Atwood, M.E. (1992) What is gained and lost when using evaluation methods other than empirical testing. In A. Monk, D. Diaper and M.D. Harrison (eds.), *People and Computers VII. Proceedings of HCI '92*, 173-201. Cambridge: Cambridge University Press.

Furniss, D. (2004) *Codifying Distributed Cognition: A Case Study of Emergency Medical Dispatch*. MSc Thesis. UCLIC.

Furniss, D. & Blandford, A. (2006) Understanding Emergency Medical Dispatch in terms of Distributed Cognition: a case study. *Ergonomics Journal*. 49. 12/13. 1174-1203.

Gong, R. and Kieras, D. (1994) A validation of the GOMS model methodology in the development of a specialized, commercial software application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating interdependence*. B. Adelson, S. Dumais, and J. Olson, Eds. ACM Press, New York, NY, 351-357.

Gray, W., John, B & Atwood, M. (1993) Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human-Computer Interaction*, 8. 237-309.

Gray, W. D. & Salzman, M. C. (1998) Damaged Merchandise? A Review of Experiments that Compare Usability Evaluation Methods. In *Human-Computer Interaction* 13.3. 203-261

Green, T.R.G. (1989) Cognitive dimensions of notations. In R. Winder and A. Sutcliffe (Eds), *People and Computers V*. Cambridge University Press.

Green, T.R.G. (1994) The cognitive dimensions of information structures. *Technical Communication*, Third Quarter 1994, 544-548.

Green, T.R.G. and Petre, M. (1996) Usability analysis of visual programming environments: a cognitive dimensions framework. *J. Visual Languages and Visual Computing* 7. 131-174.

Green, T.R.G. and Benyon, D. (1996) The skull beneath the skin: entity-relationship models of information artefacts. *International Journal of Human-Computer Studies* , 44.6. 801-828

Green, T.R.G., Blandford, A.E., Church, L., Roast, C.R. and Clarke, S. (2006) Cognitive dimensions: Achievements, new directions, and open questions, *Journal of Visual Languages & Computing*, 17.4. Ten Years of Cognitive Dimensions. 328-365.

Hertzum, M., and Jacobsen, N.E. (2001) The Evaluator Effect: A Chilling Fact about Usability Evaluation Methods.*International Journal of Human-Computer Interaction*, 13.4. 421-443.

Hollan, J. D., Hutchins, E. L. & Kirsh, D. (2000) Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on CHI*, 7.2. 174-196.

Hutchins, E. (1995) *Cognition In The Wild*. MIT Press, Cambridge, MA.

Hyde, J. K. (2002) *Multi-Modal Usability Evaluation*. PhD thesis. Middlesex University.

John, B. & Kieras, D. E. (1996) Using GOMS for user interface design and evaluation: which technique? *ACM ToCHI* 3.4. 287-319.

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.

Johnson P. (1992). *Human-Computer Interaction: Psychology, Task Analysis and Software Engineering*. London, McGraw-Hill.

Kieras, D. E. (1997) A guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, & P. Prabhu (Eds.), *Handbook of human-computer interaction* (2nd ed., pp. 733-766). Amsterdam: North-Holland.

Lewis, C, H., Poison, P. G., Wharton, C., and Rieman, J. (1990) Testing a Walkthrough Methodology for Theory-Based Design of Watk-Up-and-Use Interfaces. *Proceedings of CHI '90 Conference on Human Factors in Computer Systems*. New York Association for Computing Machinery, 235-241.

May, J., Barnard, P. & Blandford, A. (1993) Using structural descriptions of interfaces to automate the modelling of user cognition. *User Modelling and User-Adapted Interaction,* 3.1. 27-64.

Nielsen, J. (1994) Heuristic Evaluation. In J. Nielsen & R. Mack (eds.), *Usability Inspection Methods* (pp. 25-62) New York: John Wiley.

Neilsen, J. (1995) Technology Transfer of Heuristic Evaluation and Usability Inspection. Keynote paper presented at Interact 1995. Available from http://www.useit.com/papers/heuristic/learning_inspection.html (accessed 6/8/06).

Nielsen, J. and Landauer, T. K. (1993) A mathematical model of the finding of usability problems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York, NY, 206-213.

Nielsen, J. and Molich, R. (1990) Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Empowering People*. J. C. Chew and J. Whiteside, Eds. ACM Press, New York, NY, 249-256.

Pahl, G. & Beitz W. (1984) *Engineering Design,* London: The Design Council.

Payne, S. J. & Green, T. R. G. (1986) Task-Action Grammars: The Model of the Mental Representation of Task Languages. *Human-Computer Interaction*, 2, 93-133

Roast, C.R., Khazaei, B. and Siddiqi, J. (2000) Formal comparison of program modification. *IEEE Symposium on Visual Languages*, 165-171. IEEE Computer Society

Schiele, F. and Green, T. R. G. (1990) Formalisms in HCI: the case of Task-Action Grammar. In M. Harrison and H. Thimbleby (Eds) *Formal methods in Human-Computer Interaction*. Cambridge University Press.

Sears, A. (1997) Heuristic walkthroughs: finding the problems without the noise. *International Journal of Human-Computer Interaction*, 1997, 9.3. 213-234.

Sharp, H., Robinson, H., Segal, J. and Furniss, D. (2006) The Role of Story Cards and the Wall in XP teams: a distributed cognition perspective. In *Proc. Agile 2006*.

Spencer, R. (2000) The Streamlined Cognitive Walkthrough Method, Working Around Social Constraints Encountered in a Software Development Company, *Proc. CHI'2000*. 353-359.

Suchman, L. (1987) Plans and situated actions. The problem of human computer communication. Cambridge: Cambridge University Press.

Thimbleby, H. (2004) User Interface Design with Matrix Algebra, *ACM Transactions on Computer-Human Interaction*, 11.2.181–236.

Wharton, C., Rieman, J., Lewis, C. & Polson, P. (1994) The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. Mack (eds.), *Usability inspection methods*, 105-140. New York: John Wiley.

Wixon, D. (2003) Evaluating Usability Methods; Why the Current Literature Fails the Practitioner. *Interactions* July and August 2003

Woolrych, A. and Cockton, G. (2000) Assessing Heuristic Evaluation: Mind the Quality, not just Percentages. in *Proceedings of British HCI Group HCI 2000 Conference: Volume 2*, Eds. S. Turner and P. Turner, British Computer Society: London, 35-36.

Woolrych, A., and Cockton, G. (2001) Why and When Five Test Users aren't Enough, in *Proc. IHM-HCI Conference: Volume 2*, 105-108.

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.), *Wings for the Mind: CHI '89 Conference Proceedings*, pp.15-19. Reading, MA: Addison-Wesley.