Cognitive Dimensions: achievements, new directions, and open questions

T. R. G. Green[1], A. E. Blandford[2], L. Church[3], C. R. Roast[4] and S. Clarke[5]

[1] School of Computing
University of Leeds, Leeds LS2 9JT, UK
t.r.g.green@comp.leeds.ac.uk

[2] University College London Interaction Centre
UCL, Remax House, 31-32 Alfred Place, London WC1E 7DP, UK
a.blandford@cs.ucl.ac.uk

[3] Computer Laboratory, William Gates Building
University of Cambridge, 15 JJ Thomson Avenue Cambridge, CB3 0FD, UK
luke@church.name

[4] The Culture, Communication and Computing Research Institute
Sheffield Hallam University, Sheffield S1 1WB, UK
c.r.roast@shu.ac.uk

[5] Microsoft
1 Microsoft Way, Redmond, WA 98052, USA
Steven.Clarke@microsoft.com

Feb 2006. Revised for special issue of *Journal of Visual Languages and Computing*

Abstract

The Cognitive Dimensions framework has inspired research both more and less varied than expected. In this paper we revisit the original aims and briefly describe some subsequent research, to consider whether the original aims were too austere in rejecting knowledge-based dimensions; whether the dimensions can be shown to have real-world relevance; and whether their definitions can be improved, either piecemeal or by refactoring the entire set. We mention some issues that remain unexplored, and conclude by describing two different ventures into defining clear procedures for real-life application, operating in very different milieux but both accepting that the framework should be developed from its original formulation.

1. Introduction

Brain-children, like other children, are inclined to pursue their own course, and their progenitor can like it or lump it. The intention of this paper is to consider how the Cognitive Dimensions framework, the topic of this special issue, has met the original expectations, where it might be heading, and what, if anything, can be done to patch up the holes in the road it has traversed so far.

Let us start with a reminder of the original intentions. After an introduction claiming to offer "a generalisation which is not yet fully worked out but promises, if successful, to be powerful", Green [1] concludes at the end of the paper that:

> "A successful scheme of cognitive dimensions would bring many advantages. First, description of relevant features of designs would be more compact, less ambiguous. We can, in future, refer to a particular notation as being, say, 'viscous', rather than having to spell out the point in laborious English: 'If you want to change something, it's a lot of work to undo what you've already done and modify it'.
>
> Second, the analysis turns out to clarify an ill-understood issue: how particular environmental features complement the notation. … We can also make better comparisons of trade-off issues between different notational features. …
>
> Third, it becomes clear that auxiliary notations have an important role to play, especially in programming tasks (informal notations, intermediate representations, program design languages, programming methodologies, etc.): they function by allowing the user to adopt the preferred cognitive strategy even in apparently adverse circumstances." [1]

In later formulations, some of these advantages were summarised as 'discourse tools':

> "In other words, I am trying to provide *discussion tools*, to raise the level of discourse among choosers and users who are domain specialists but who are not computer scientists, HCI specialists, cognitive psychologists, etc. By providing terms to be used in discussion, which together make up a short check list, it will be easier to converse without having to explain what one means at every step; also, it will be easier to understand the trade-offs between different structural features. These, of course, are just the familiar advantages of a discourse based on a shared understanding of concepts and terms. What I am hoping to do is to create that shared understanding." [2]

Thus, the original intention was to improve design practice by making it easier to talk about design usability at an appropriate level of abstraction.

In the first formulation of the framework [1] only one type of activity was considered, exploratory design. The paper proceeded by considering the phenomena of exploratory design, notably the well-evidenced occurrence of redesign events: at any point during the design process, the current version might be changed, usually at a local level but sometimes at a much deeper level. Ease of redesign imposed certain requirements on the system, such as low viscosity – a concept further explored by Green [3].

Soon the list of dimensions had grown considerably. There was no systematic procedure for generating dimensions: rather, members of the team (including Rachel Bellamy, David Gilmore, David Hendry and Marian Petre, among others) would be alert for personal experiences or reported findings that could not readily be captured by the existing dimensions list, and the list would be extended when necessary. By the time of the first heavyweight application of the framework [4], in which a selected group of visual programming languages (VPLs) was evaluated, we were pleased to find that most of our personal experience of using those languages and others was captured by those dimensions, and that formal laboratory experimentation on VPLs brought no surprises [5].

Already, although we did not know it, the framework was about to leave the intended path. We had intended the framework to apply not just to visual programming languages, nor even to the more extensive field of all programming languages, but to all information-based artefacts, from whatever domain. Green and Petre applied them to visual programming languages because VPLs are interesting in their own right, because they were exciting some degree of research activity, and because the field was small enough for a few selected VPLs to be reasonably representative. The conclusion unfortunately drawn by some readers was that CDs were a method for evaluating VPLs and only VPLs.

The version of the framework used by Green and Petre [4] was still focused on exploratory programming, but the importance of other activities was beginning to become clear. At the time of creating a CDs tutorial [6] a reasonable range of activities could be listed, and a table of the relevance of each dimension to each activity could be presented. Further suggested dimensions have emerged since then, discussed in section 2.4 below. As with all other developments in this framework, the support for the list of activities and the list of dimensions was, and remains, extremely informal.

The degree of informality was a cause for concern, since it seemed to lead to misunderstandings and even to deliberate reshaping of the dimensions. It triggered several attempts to improve the precision of definition. Green [7] made the first attempt, using a version of entity-relationship modelling to show how the structure of an information artefact determined certain aspects of cognitive dimensions, and this was greatly extended by Green and Benyon's development of ERMIA [9] [10], an explicit formalism for Entity-Relationship Modelling of Information Artefacts. ERMIA had considerable success in some ways but it, too, displayed the wantonness with which ideas leave their intended courses: we chose the formalism because its diagrammatic roots in entity-relationship modelling seemed likely to allow quick, lightweight analyses of artefacts, but we found eventually that the diagrams were too complex to be easily comprehended, and were not so easy to generate as we had hoped. In other words, Green and Benyon had overshot the mark. Using the Investment of Attention model, a very different approach in which attention is treated as a scarce resource to be husbanded not squandered, Blackwell and Green [11] analysed some of the decisions facing document users, finding clearer distinctions between activities than had previously been set down—but, in the end, providing more of a descriptive model rather than a prescriptive one.

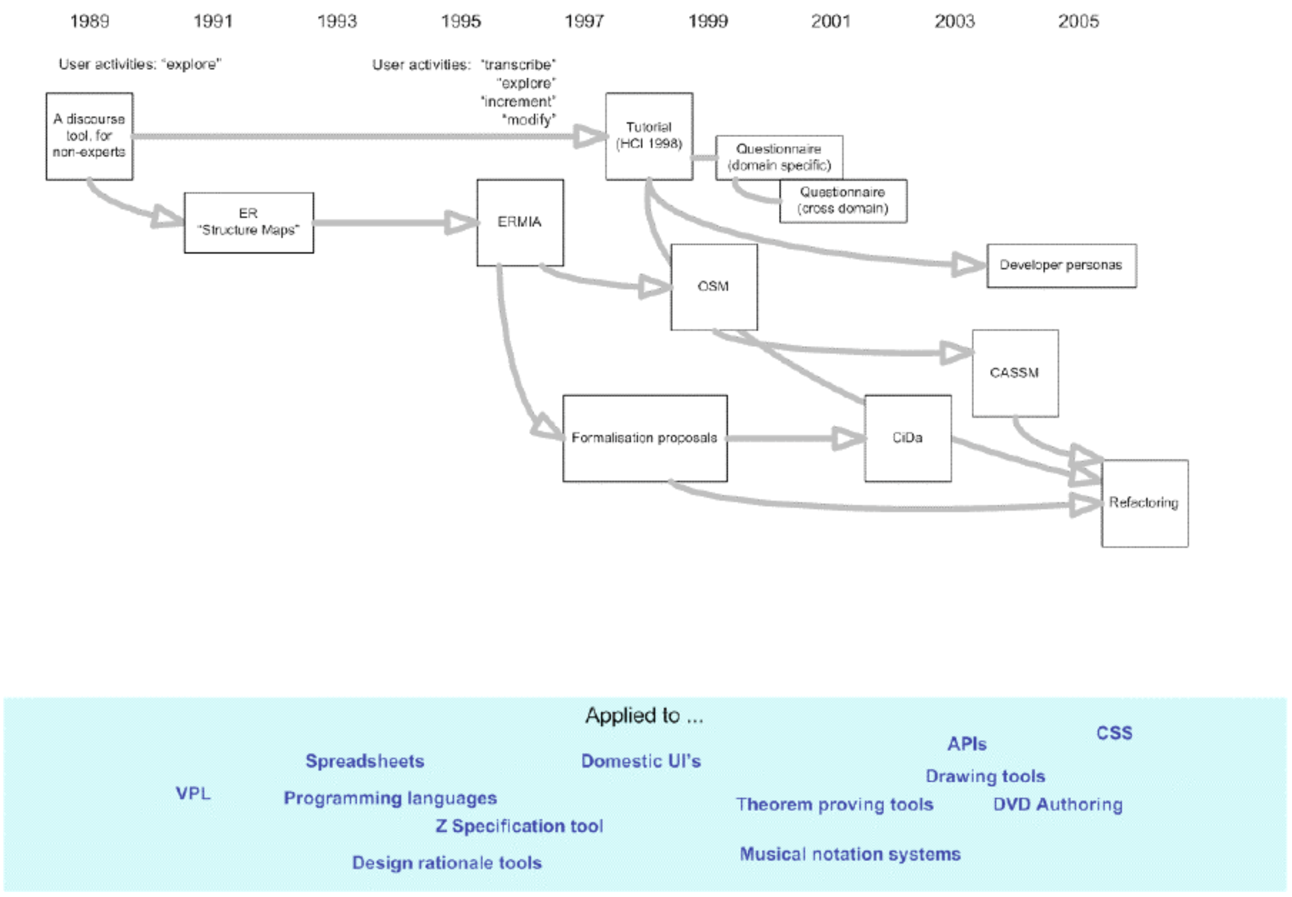


Figure 1: Time-line for initial developments

Real-world demands also affect the procedural aspect of the cognitive dimensions framework: insofar as it had a procedural component at all, the original conception was that the analyst understood the domain of analysis and was familiar with the notations and tools to be analysed; given which, producing an analysis was a short job. In the real world that may well not be the case. Kadoda [12] [13] made the first step towards a procedure that did not require the analyst to be a domain expert (and at the same time was more systematic than previous procedures) by devising a cognitive dimensions questionnaire. Her questionnaire was specific to the domain of analysis, which in her case was computer-based theorem-proving. Blackwell and Green [14] extended that approach by attempting to devise a questionnaire that was general and non-specific. Their success was no more than partial, since the resulting questions tended to be too abstract for easy comprehension; nevertheless, this line of development showed that it would be helpful to move towards a more standardised and better-specified procedure, possibly with a standardised set of questions to ask. Further questionnaires have been developed, and here again the original idea can be perceived to have leanings of its own, since many devisers of questionnaires have chosen to work with a subset of the list of dimensions, arguing that the other dimensions are not relevant in their domain.

Not surprisingly, the informality of definition has led to alternative interpretations of the dimensions. Different researchers and designers have formed slightly different understandings of the sketchy definitions; even when understandings are shared it is frequently a matter for debate as to how to interpret a given example. Moreover, the informality also makes it impossible to know how far the list is comprehensive. Merely adding a new dimension when it was found to be needed, as has regularly been done, is no guarantee that enough dimensions exist to cover the important cases; and when new dimensions are added by other researchers, there is an additional opportunity for the idea to leave its intended path, for other researchers may well have a different conception of what it means to be a *cognitive* dimension. They may even be forced by real-world demands to introduce dimensions that are recognisably non-cognitive.

This little history, brief though it is, is already sufficient to reveal some of the unexpected, and sometimes dark, corners into which the CDs framework has wandered.

The remainder of this paper will consider some of these issues in more detail:

What is the basis for the existing dimensions?

Do we have any reason to suppose that the framework has real-life relevance?

Can the method of using the framework be made clearer and easier, especially for non-specialists?

Is there any prospect of reconstructing the dimension set, to be more precise, with less overlap, while remaining easy to use?

What areas remain dark corners?

Where does all that leave us?

2. What is the basis for the dimensions?

As proposed originally, the CDs framework was intended to meet two major criteria. First, everything in it should be entailed by a reasonable cognitive analysis of the activity being pursued; and second, it should reveal the points of deep similarity between devices or notations, possibly drawn from quite different domains. An undeclared aim was that all dimensions should be reasonably well defined and should be fairly easy to understand; and, naturally, the hope was that the original set would be more or less complete and would be a useful tool in practice. We discuss each of these in turn, noting in particular that the aim of restricting the framework to 'deep similarity' has come under pressure from the push for completeness and practical application.

2.1. Cognitive analysis of activities

The first criterion is that everything in the framework should be entailed by a reasonable cognitive model. In the original version of the CDs framework only one activity was considered, exploratory design. In the world of software design, folk psychology had originally produced the now discredited 'waterfall' or 'top-down' model of design, propounded enthusiastically by many a pundit in many disciplines (e.g. [15]). The process was divided into stages: high-level decisions, medium-level, low-level, as many stages as took the pundit's fancy. At each stage, the designer was to solve all the problems particular to that stage, then go on to the next stage. No earlier stages would ever need to be revisited.

Observations of designers at work, cited in [1], showed little conformity to the waterfall model. Designers work at different levels of abstraction and detail concurrently, rather than proceeding from higher to lower, and they do not follow a balanced development strategy; they recognise opportunities or problems and shift attention as needed, and at any moment they may tinker with details or with deep decisions. In a memorable slogan of the 90's, "design is *re*design". A number of more plausible models were developed, such as the 'top-down with variations' model [16].

If the notation and the environment together are to support such behaviour by designers, then at least we can say that it must be possible to *perceive the structure* of whatever is being created; that the structure must be *readily modifiable*; and that since early stages of the design represent a wide band of possible developments, the notation should differentiate between degrees of definiteness by indicating *provisionality*. All the proposed dimensions should have their place in this scheme.

The further types of activity that have been introduced to the framework have been prompted by considerations of what users do with information, apart from create it. Such activities have included modification of structure; incrementation; transcription; search; and exploratory understanding. The details of these activities, and especially the relationship between the activity and the notation plus environment, have been less well modelled than the design activity, so far as the authors are aware, but it has not proved necessary (so far!) to modify the framework greatly in order to accommodate them. So we are left with the following goal: *in the well-developed framework, each of the cognitive dimensions should be justifiable by pointing to its place in a cognitive model of at least one of the types of activity that the framework seeks to cover*.

2.2. Revealing 'deep' similarities

This is the second of the two criteria introduced above. To illustrate it, sentence structures offer a useful parallel. The structure of the sentence "sheep may safely graze" remains the same whether it is spoken by one person or another person, or even written down; and that structure remains the same even if the individual words are changed into "rhinoceroses may lugubriously cogitate". Obviously, many features do change when the sentence is differently spoken, is written instead of spoken, or is reworded; but in order to define the grammar of a language, it is important to have a concept of what does not change.

Consider some simple device, such as an online MP3 player, with a small set of controls. Typically, online MP3 players are 'skinnable', meaning that a third party can devise a new look for the device (Figure 2). Because the fundamental operation of the device is unchanged by reskinning, we want an evaluation method that will be *invariant across changes of appearance*. The requirement of invariance rules out the various HCI evaluation techniques that measure whether the device is physically harder to operate in one skin than another, by counting the number of keystrokes or the accuracy of aiming that is required, as in methods derived from the Model Human Processor [17]. The invariance requirement also rules out such measures as whether the device "uses the user's language", as in Heuristic Evaluation [18]. That is not to say that such measures have no role, just that there are roles they cannot serve.

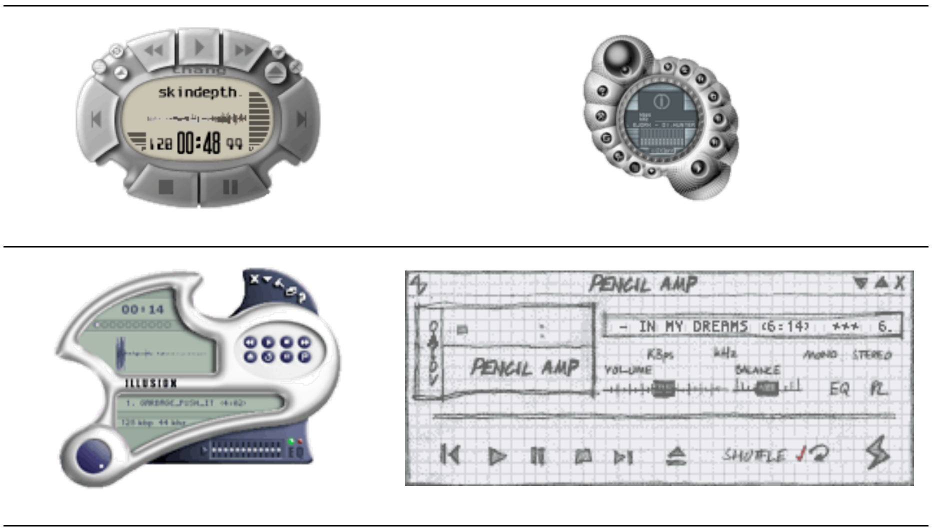


Figure 2: Alternative skins for two on-line MP3 players, the Kjofol player (upper row) and the Winamp player (lower row) [19]

Better still, we want an evaluation method that will continue to be invariant if the same device is reconstrued as something quite different—for instance, ordering meals in a restaurant has much in common with playing sound files on a computer, in terms of defining a sequence of items to be delivered in the near future, and we can imagine devices that are internally very similar. We want an evaluation method that will be *invariant across changes of domain*. That rules out knowledge-based evaluation techniques, such as Task Knowledge Structures [20]. Again, it is not that knowledge-based measures have no role, just that there are roles they cannot serve.

An invariant evaluation method is obtained if we limit ourselves to the structure of the notation, the environmental tools for exploring and altering that structure, and the human cognitive functioning. With such an evaluation method, one that is invariant across changes of domain and changes of rendering (skin), we can hope to find that we have a vocabulary of terms that can be helpfully deployed to evaluate any and every system, allowing us to make meaningful comparisons.

Subsequent developments—the use of the CDs framework by other researchers with other priorities—have pushed against the original conception, and have shown that it was perhaps too austere for practical use, especially in its rejection of domain knowledge.

2.3. Clarity, precision, exposition and utility: dimensions of dimensions

The path of the CDs framework over the years has been marked by changes in the personnel of the dimensions list. Moreover, some of the dimensions have proved to be much more popular than others. When Blackwell (personal communication) informally asked computer science students to comment on a proposed design from a CDs perspective, without prompting any particular dimension, he found that although 15 dimensions received at least one mention and 122 mentions were made overall, three-quarters of the mentions were to just four dimensions: viscosity, visibility, hidden dependencies and premature commitment.

Some dimensions have been given little attention because they seem to be irrelevant in a particular context. To someone evaluating a menu-driven system, viscosity might seem irrelevant. Much more disturbingly, some of the original dimensions have turned out to be hard to understand or have been poorly described. Examples here are the ‘hard mental operations’ and the ‘role-expressiveness’ dimensions, which have always been poorly described; and two others have perhaps never been properly described, even though mentioned in the original paper [1]: ‘perceptual cues to structure’ and ‘discriminability’.

The *hard mental operations* dimension (HMOs) was intended to cover overload in the cognitive processing. There are well-known cases where small examples are easy, even facile, but where the difficulty rises explosively as the examples increase in size. (In retrospect it would have been better to call it something like ‘potentially-explosive mental processes’—where ‘explosion’ is being used in the same sense as in a combinatorial explosion). Linguistic examples include multiple negatives and self-embeddings:

> *Nobody doesn’t have to not go unless they don’t want to not go.*
> *The goose that the cat that the flea bit chased honked.*

There are also puzzles of the ‘knights and knaves’ type:

> *'You meet two inhabitants: Sue and Bart. Sue tells you, "Bart could claim that it is not the case that either I am a knave or Bart is a knave." Bart tells you, "I am a knave and if Sue is a knight then both I am a knave and Sue and I are both knights or both knaves."' Which of them is a knight (and therefore always tells the truth) and which is a knave (and therefore always lies)? [21]*

These linguistic and logic-based difficulties seem so obvious that one might wonder whether this is a contrived dimension, but Green and Petre [4] identified a striking example in the design of one VPL, even though it was presented as being easy to use. This VPL, ‘Show and Tell’ [22] uses a dataflow model in which the usual conception of conditionality has been replaced by a consistency constraint: data cannot flow into or through a box that is in an inconsistent state. Inconsistency originates when a box receives two or more inputs that do not agree, or when its input disagrees with a stored constant. Single-box diagrams are easy to understand but Green and Petre reckoned that diagrams using multiple boxes, some contained in other boxes, were very taxing indeed.

Hard mental operations can also cause overload by spawning too many subprocesses, such as searching a maze with no external aids—and again, it is possible to find examples in the design of VPLs, as Green et al

[5] demonstrated empirically. Similarly it is arguable that some of the notorious novice difficulties with pointers in languages like C arise because dereferencing the structure part by part is just too difficult for novices; they have to learn to treat it as an idiom or a pattern, focusing on what it does as a whole. Hence there are tutorials, such as [23], claiming that "Using pointers is a bit like riding a bicycle. Just when you think that you'll never understand them—suddenly you do!" Such tutorials emphasize typical--even stereotyped--patterns of usage, rather than emphasizing reasoning about contorted examples.

At an informal level it is not difficult to offer a cognitive processing model (which is, of course, completely untested). All the examples have these two features:

(1) the operation is easy to do once (a single negative is trivial), ok to do twice (a double negative is fair), but explosively hard to do more than twice

(2) and when the person has to perform the mental operation more than once, the input to the Nth instance is the output from the (N-1)th.

A plausible starting hypothesis is simply that the chunks in memory are liable to get mixed up. To do the operation once, no memory problem is created. To do it twice, the output from the first has to be kept available as input for the second. To do it 3 times, then on the third operation, there are two contenders for input, namely the output from the first and the output from the second. At this point—but crucially, not before—there is now competition between two chunks in memory, either of which might be the correct input for the third instance of the operation; introducing time delays while the correct chunk is identified, and frequent errors when the wrong chunk is chosen.

The *role-expressiveness* dimension has also been problematic. This is, we suggest for an entirely different reason. The hard mental operation, once understood, is not hard to spot, and can be adequately, if informally, modelled. Role-expressiveness, in contrast, is irredeemably vague. The essential test is whether the purpose of each component in a structure can readily be discerned. It is not possible to make a simple cognitive model, and so it is not easy to do more than present examples that appear to illustrate good and bad role-expressiveness. Hardly surprisingly, users of the CDs framework can be hesitant in using something so laxly defined. It is also (we thank the Guest editor for reminding us) readily confused with *closeness of mapping*, although the concepts are distinct. Closeness of mapping refers to the conceptual distance between an outcome, and the action or instruction required to achieve that outcome, whereas role-expressiveness refers to the purpose of an outcome. One can see, with hindsight, that role-expressiveness can readily be misconstrued as the purpose of an action or instruction—which makes it effectively the same as closeness of mapping. Perhaps these two should be restated and renamed!

The presence or absence of *perceptual cues to structure* (briefly mentioned in [1]) is related to role-expressiveness. Both concepts originated in work on program comprehension. Using a command-based line editor, Payne et al [24] showed that certain kinds of mistakes could be reduced by the simple means of putting commands in capital letters, allowing perceptual parsing of structure; Gilmore and Green [25] showed that if each hypothesized mental chunk in a small Pascal program was differently coloured, certain types of bugs were more readily spotted. In the same vein, but drawing a different kind of evidence, Green and Borning [26] showed that a plausible model of human parsing, extended to cover program code, had more difficulty parsing Prolog than Pascal, because Pascal uses lexical indicators for such constructs as iteration while Prolog relies on the shape of the rule and the placing of the arguments. The literature of information visualization and diagram design is no doubt filled with similar findings, tending to show that comprehension is easier when important structural differences are mirrored by perceptual cues.

Unfortunately, as further examples are collected, it becomes clear that 'perceptual cues' lie somewhere between role-expressiveness and simple discriminability. Discriminability—or rather its opposite, indiscriminability—is included in [1] as a contributor to error-proneness. In the Green and Blackwell tutorial [6] the example is from the programming language Forth, which uses the symbols ',' and '.' (comma and period) as two quite different commands; likewise, Fortran encouraged the letter I to be used as an identifier, hardly distinguishable from numeral 1 in some fonts. Not much evidence is needed to underline this one. But one has to ask: is ‘role-expressiveness’ just an issue of discriminability at the level of structures rather than lexemes?

2.4. The completeness issue

Although the original thought was that the list of dimensions would be more or less complete, many further dimensions have been proposed. One of the first additions was secondary notation: observations of professional dsigners made it clear that they needed to more than produce a design—they needed to have some way to represent further information, information that was *about* the design rather than part of it [8].

There are at least two tensions in propsing new dimensions. First, a genuinely complete account of all the cognitive factors relevant to interaction with information devices would be far beyond present knowledge, and in any case would be quite unusable. We need to identify, as cognitive dimensions, factors that make a substantial difference to usability in a large number of cases. Second, we need to decide whether to maintain the original aim of rejecting both the user’s domain knowledge and the surface representation of the information structure.

Many of the newly-proposed dimensions have tended to lessen the austerity of the original conception, especially by introducing a knowledge-based component. One such example is a proposed dimension originally called ‘tunability’ **Error! Reference source not found.** but which we shall here dub ‘penetrability’, where a penetrable system is one that supports the user in how the internals of the system work at an operational level. This is of specific benefit to users where knowledge of the internal characteristics of a system can enable the effective use of the system.

Creative Ambiguity

The extent to which a notation encourages or enables the user to see something different when looking at it a second time

Specificity

The notation uses elements that have a limited number of potential meanings (irrespective of their defined meaning in this notation), rather than a wide range of conventional uses

Detail in context

It is possible to see how elements relate to others within the same notational layer (rather than to elements in other layers, which is role expressiveness), and it is possible to move between them with sensible transitions, such as Fisheye views

Indexing

The notation includes elements to help the user find specific parts.

Synopsie

(originally “grokkiness”) The notation provides a gestalt-like understanding of the whole when you “stand back and look”.

Free rides

New information is generated as a result of following the notational rules

Figure 3: A summary of suggested new dimensions in ***Error! Reference source not found.*** *- see that paper for background sources*

These inferred workings correspond to an operational model of the system—the ‘how it works’ model – as opposed to the conceptual model, the ‘what it does’ model. The user’s conceptual model can be in opposition to that envisaged or planned by the designer. In these circumstances the user encounters problems, and may resort to using an operational model. If the operational model is wrong, the user will have problems. Empirical demonstrations of such plausible, but incorrect, inference have been given by Cox and Young [30], who observed people trying to understand a novel system for domestic central heating control.

Non-technical users' interpretations of web browser behaviour often reveal incorrect operational accounts that are fully conformant with their experiences; for example, one can believe that a hyperlink marked as 'followed' shows the target page is in fact locally cached. If locally-cached pages were also shown in a manner distinct from those followed, then the belief would be less likely to be entertained: the system would have moved to a higher position on penetrability.

Nor need these incorrect but usable operational models be confined to novices. Payne [31] quizzed experienced users of word-processors and found that their concepts were surprisingly vague and inaccurate. In practice, they relied on display-based problem-solving far more than on the possession of an accurate internalized model.

But does this proposed new dimension fit the original conception of 'cognitive' dimensions? Not entirely; it extends the original conception: and to draw attention to the way in which it extends the conception, we take an example from the Discworld novels of Terry Pratchett, in one of which we find a camera that turns out to contain a tiny demon with a pot of paints—an acceptable, if unlikely, operational model of a pre-digital camera.

> "There was a time when Rincewind had quite liked the iconoscope. He believed, against all experience, that the world was fundamentally understandable, and that if he could only equip himself with the right mental toolbox he could take the back off and see how it worked. He was, of course, dead wrong. The iconoscope didn't take pictures by letting light fall onto specially treated paper, as he had surmised, but by the far simpler method of imprisoning a small demon with a good eye for colour and a speedy hand with a paintbrush. He had been very upset to find that out." [32] p. 129

The example is not presented for pure whimsy. Readers will realise that this is an incorrect model of how their own camera works; but they will only be able to do so because they have extensive *knowledge* about the world they inhabit, where tiny demons with pots of paint have not featured. In other words, this example shows that the plausibility of an inferred model depends on background knowledge, as well as on the 'transparency' of the system. A knowledge component is exactly what the original conception of the CDs framework was intended to avoid, and this example illustrates how subsequent developments have pushed against that austerity.

Perhaps it is time to relax the austerity of the original conception; or perhaps that would eventually lead to a witch's brew into which anything could be thrown, with little or no theoretical justification. Such a brew would moreover contain far too many ingredients. A framework of a dozen-plus dimensions is already too large for some practical purposes, even though, as noted in the introduction, each had been arrived at in order to capture something observed or reported.

There is no real way to assert completeness of the dimensions list, as far as we know. Indeed, the original paper suggested that known completeness was unattainable:

> "The physicists' dimensions form a closed set. Do these cognitive dimensions form a closed set, or are they just plucked from the air and from experts' insights, like guidelines? I suggest that the closest analogy may actually be with concepts taken from biology, such as plant growth. The mechanism of plant growth is extremely complex, but we know that normal green plants need phosphates, nitrogen, sunlight, etc., to grow successfully. We also know something about how plants respond to different combinations. Best of all, we understand a good deal about the mechanism of plant growth, so that we can make many predictive assertions. What plant biologists do *not* have, and cannot have, is a logically complete closure of orthogonal concepts, analogous to the physicists' dimensions. There may always be another trace substance, undetected as yet, required for vigorous growth." [1]

3. Do we have any reason to suppose that the framework has real-life relevance?

There are a good number of techniques within HCI for evaluating interaction design: how successful these techniques are is hard to gauge—Gray & Salzman [33] have trenchantly reviewed evidence derived from a number of experiments designed to compare the efficacy of different evaluation techniques, and have made it clear that this is no easy issue to address. But with regard to CDs and their real-world relevance we are in a fortunate position, able to finesse the whole problem of experimental design. Instead of inferring putative real-world relevance from lab studies, sometimes dubiously and always optimistically, we can go straight to where we want to be: we can look directly at real-world developments.

We can do so because there is a domain that historically has a fairly well-defined starting point, such that we know the notation and the original environment of use; that is suitable for the application of CDs analysis to reveal possible problems; and (this is crucial) is lightweight. Because it is lightweight, a wide range of real-life 'design manoeuvres' have been developed to ameliorate its problems. Thus, if there is a close relationship between the design manoeuvres observed and the problems predicted, it follows that the CDs analysis did a good job. Ideally, each predicted difficulty will be associated with a design manoeuvre intended to lessen it, showing that each prediction has validity; and each design manoeuvre observed will relate to a predicted difficulty, showing that the set of predictions is fairly complete. If we encounter design manoeuvres relating to *unpredicted* sources of difficulty, that will indicate gaps in the coverage given by CDs. We can be still more accurate by pointing out that a design manoeuvre need not be a new environment or a change in the notation; it can also be a change in the work system that gets round the problem. And where the problem is intractable, real-world evidence of continuing difficulties will also count as helping to verify the validity of CDs predictions.

The domain in question is that of hypertext mark-up language (HTML) and Cascading Style Sheets (CSS), the notations commonly used for building websites. The notation of HTML is small, in the sense that a few generalisations (meta-rules) are sufficient to describe most of the structure. The original environment of use was the humble text-editor, at least for many early adopters, and the possible problems can readily be analysed. Likewise the notation of CSS is small. It is much easier to build environments for HTML and CSS than to build a full-scale IDE for a programming language like Java, and with the number of websites increasing exponentially the demand for such tools is great; so there are a good few tools to look at.

3.1. *The structure of HTML websites*

Although this will undoubtedly be familiar ground to many readers it is worthwhile to explain the basic features of HTML. The later development of CSS will be covered in the following section. Text files marked up in HTML code and made available to browsers (web pages) contain a head part, which need not bother us in this context, and a body part. The mark up in the body part uses tags, which mostly have an opening part and a closing part, such as the tag for a top-level heading, `<h1> ... </h1>`. Text between those tags will be displayed in whatever style that browser chooses for a top-level heading, typically in a large bold font face. Individual pieces of text can be given stylistic attributes by setting the font size, font family, and font colour; so `<p><font size="+2"> lorem ipsum ... </font></p>` specifies a paragraph to be displayed two sizes larger than default. (This usage is now deprecated in favour of CSS styling, described below, but remains usable, and is widespread across the web.) References to other web pages are made by an anchor tag, with the basic structure `<a href="URL"> See URL </a>`, which provides a hyperlink: the text displayed is 'See URL', and if that text is clicked the file at web address URL is displayed. There is no abstraction mechanism in the syntax, but there is a comment syntax. Typically a website contains a large number of linked files, and a particularly common structure is for each top-level page on a website to contain a link to each other top-level page, usually arranged contiguously on the page as a navigation bar.

Assuming that this text is being prepared in a simple text editor, we can immediately make at least a superficial CDs analysis.

- Premature commitment is low, at least within the individual page, but is not entirely absent, since it may turn out for example that the text styling chosen at the start of the page design is not suitable.
- Viscosity, on the other hand, is rather high, both within the individual pages and within the site. Within the individual page, a decision to restyle all level 1 headings requires each such heading to be visited and restyled individually. At the site level, the navigation-bar structure creates massive viscosity: adding or deleting a top-level page, or changing the name of an existing page, requires every existing top-level page to be visited to have its navigation bar changed.
- Progressive evaluation is rather poor, since the page must be rendered in a browser before the effect of the HTML mark up is apparent. Indeed, different browsers give different renderings, so ideally the page must be rendered in several browsers, not just one.
- There are no hidden dependencies at page level, but the links create hidden dependencies at the site level, because it is not easy to know which pages contain links to a given target page.
- Visibility is good, secondary notation is available through the comment syntax, and the consistency of HTML mark up is very high.
- Hard mental operations do not arise: there are no long search trails to be remembered and embedded structures, although pervasive, are not hard to parse.
- The abstraction level is zero: – this is an abstraction-hating system.
- Closeness of mapping is tolerable for much of the mark up scheme (e.g. ‘h1’ for ‘header’) but is less good for lists and definitely poor for links, which use the ‘a’ tag described above. Even for the best cases the mapping between a tag <h1> and the rendering of a header is not very good.
- Role expressiveness presents few problems.
- Perceptual parsing is rather poor, but luckily the internal structures of HTML code are not complex.
- Lastly, the mark up must be regarded as fairly diffuse for certain structures, notably for tables. The mark up for a simple 2 x 2 table comprises two tags for the table (start and finish tags), two for each row, and two for each individual cell, a total of 2+4+8=14 tags: all this to lay out four content elements.

As an exploratory design system, HTML in a simple text-editor clearly fails to meet the requirements. Redesign, an example of a modification activity, whether at the site or the page level, runs into heavy viscosity difficulties, as well as the lack of progressive evaluation. Likewise incrementation is problematic at the site level, because of the work required to update the navigation bar. Even as a transcription system HTML has problems, because the lack of abstractions makes it difficult to achieve true uniformity of design across pages or sometimes even within a page.

Early developments included environments designed to increase progressive evaluation by presenting the rendering side by side with the mark up. Some, such as the well-known Macromedia Dreamweaver [34], allowed a degree of direct-manipulation WYSIWYG-style editing, in which familiar word-processor-like keyboard commands could be used: for example, pressing the Return key created the HTML mark up for closing one paragraph and starting another.

The diffuseness of HTML code, not to mention the poor closeness of mapping, was also improved by such WYSIWYG environments. The number of actions required to create a table or to modify its size or other properties was greatly reduced by using menu-driven operations. In addition, the notation itself was modified in some systems, especially environments like blogs (web-based logs) and wikis (informal, reader-editable encyclopaedias) in which succinctness and immediacy of response was valued. An interpolated front-end allowed users to input lightly-structured text, and simple rules transform that text into HTML code; so a list might be input in this form:

```
- List item 1
- List item 2
```

```
        - List item 3
```
generating the following HTML code:

```
        <ul>
              <li>List item 1</li>
              <li>List item 2</li>
              <li>List item 3</li>
        </ul>
```
This tactic is an example of ‘decoupling’, one of the work-around manoeuvres listed in the tutorial [6], in which an intermediate notation is used for preliminary stages: a typical example is using a pseudo-code or some other sketchy notation for drafting a program, then translating that into the final target code. There is an obvious trade-off here, in that the user now has a further notation to learn, but in return can escape the constraints of the final target notation.

Viscosity was the other dimension that first-generation coding environments sought to assuage. At the site level, the viscosity associated with moving and re-naming files was reduced by introducing a ‘files window’; operations within that window simultaneously operated on the files themselves, and upon all links referencing that file. For a large site the saving was very great at times: the whole folder hierarchy could be re-organised quite simply. Perhaps less successfully, some of the environments, such as Dreamweaver and BBEdit [35], introduced a notion of templates, specifying some of the components of a page—for example, the set of links comprising a navigation bar could be made into a template. When a new link needed to be introduced, the template could be edited and then the site as a whole would be updated automatically. Templates were distinguished in the HTML code by syntactically-specific comments, essentially saying “this is the start of a template region taken from file FFF” and “this is the end of the template region”. Clearly, templates are a form of abstraction mechanism bolted onto the original, abstractionless notation by using some of the space of secondary notation possibilities.

The template scheme well illustrates one of the standard trade-off issues. The template has to be set up early in the design process, probably before the designer has got the details clear. In other words, the designer will have to accept some degree of premature commitment in defining the template. Furthermore, the process of defining the template places its own load on the designer’s attention, and he or she must decide whether or not to invest attention in that way.

3.2. The addition of CSS

Not mentioned so far have been the problems incurred by the absence of an abstraction mechanism, causing lack of uniformity across and within pages, and causing considerable viscosity in modifying style attributes. The response was to create Cascading Style Sheets (CSS), similar to, but considerably more powerful than, the style sheets used in word processors. Each style sheet consists of a set of rules associating a selector, such as ‘p’, with a set of rendering or positioning attributes. The simplest attributes determine font size, etc; for example, it is possible to specify the font-family and size of every paragraph on a page by the following simple rule:

```
        p { font-family: “Lucida Grande”; font-size: 16px; }
```
Using this mechanism, all the built-in tags for paragraphs, lists and headers could be made uniform within the page. Furthermore, the styles could be assembled into one or more separate files, and each page could link to the style files, so that a change in the definition of a style would take effect across an entire site. The viscosity of the HTML code was thereby dramatically reduced. (It must be said that CSS is considerably more powerful than we have room to describe. This power was a response to needs foreseen from the earliest days of its conception, as described in an elegant and beautiful account by Lie and Bos [36], two of the creators.)

From a CDs perspective, with the introduction of abstraction mechanisms comes a new layer of notation, in which the abstractions are expressed (CSS in this case), and the need for a subdevice to handle the creation and modification of those abstractions. We will initially assume the basic text-editor as that subdevice and consider the properties of the new, joint system, HTML/CSS. First, within the CSS notation itself, the CDs analysis is very similar to the analysis of HTML code. There is low premature commitment and poor progressive evaluation, as in HTML; visibility is good, a secondary notation is available via a commenting mechanism, and surface-level consistency is excellent (but see below). Perceptual parsing is offered both by indenting mechanisms and by syntax colouring, available on almost all specialised CSS editors. Hard mental operations are not a serious issue. Like the original HTML, CSS is unfriendly to abstractions: there is no way to group related rules even when they share common features, except by making use of the inheritance system, which creates problems of hidden dependencies. Inheritance uses a box model: many of the HTML tags, such as `<body>`, `<p>` (paragraph), `<h1>` (heading) and `<div>` (division), create a new layout box in the rendering, and all mark up between the opening and closing tags is considered, by default, to be contained in that box. Elements within a box normally inherit many of their properties, such as font sizes, from the properties that apply to the containing box. Thus if the CSS file includes a rule setting the font family at the body level, that rule will apply by default to all text on the page, since all components are included within the body. In practice, a given component may lie within, and therefore be influenced by, many different containers, and determining the specific style attributes may not be easy; working in the opposite direction, determining what elements on the page are affected by a given CSS rule can only be done by exhaustive inspection. These are the classic phenomena of hidden dependencies.

The CSS rules also introduced some problems of consistency. We shall take note of two examples. The first, and easiest to describe, is that elements within a containing box do not inherit *all* their properties by default from that container. Font-size, as mentioned, is inherited by default; but margins are *not* inherited. This fact is entirely consistent with the conceptual model of the designers of CSS but novices can readily trip over it. The second example is lengthier to explain. A container box, such as a `div`, normally expands to contain all the elements contained within it; but there is a CSS mechanism that takes elements out of the normal container flow, and in those cases the `div` will not necessarily expand around those elements. Within the conceptual model of the creators of CSS, this is perfectly consistent, but it is clear that to the inexpert user some such cases appear inconsistent and cause nasty surprises. A typical example, called the ‘drooping flag’ phenomenon, occurs when an image is placed inside a `div` box, with mark up like the following:

```
<div>
      <img ……> [code for inserting an image]
      <p> Lorem ipsum . . . </p>
</div>
```

If no CSS rules have been applied to take elements out of the normal flow, the `div` will surround both the text and the image, as expected. But if the CSS contains a rule to float the image to one side, the result may well be that the image hangs out of the bottom of the box (hence the ‘drooping flag’ sobriquet); the `div` expands horizontally to surround the text, but does not expand downwards. To the unsuspecting, the result is inexplicable. In Figure 4 the upper figure, (a), shows a `div` box surrounding an image and a short piece of text. In (b), we have added one CSS rule: `img { float: left; }`. That rule does not appear to do anything except change the relative layout of the image and the text—the fact that it removes the image from the normal flow is not at all apparent. Consternation results.

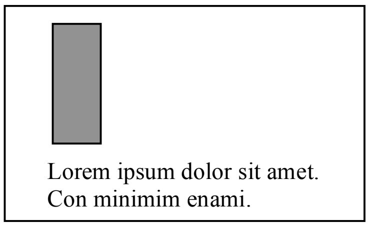
The ‘drooping flag’ may be remedied, as in (c), by adding another element that is specified not to allow anything beside it. That element need not be visible, and the following rules specify a div element that is clear on both sides and is 1 pixel high.

```
CSS:
.clearing { clear: both; height: 1px; }
HTML:
<div class = “clearing”> </div>
```

Regrettably, this trick creates a component whose role is hard to discern. We have here an element in the HTML code that has no content and exists solely to invoke the CSS formatting rule; and in the CSS rules, we have a rule to specify elements that have nothing alongside them. No part of the CSS rule explains why it exists or what its context of use is. Role expressiveness for this trick (and several others in CSS) is seemingly very poor.

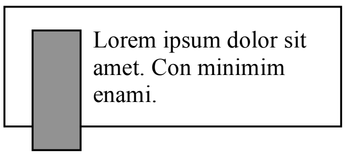
CSS rules | HTML markup | result

```
<div class="bordered">
   <img src="flag.gif">
   <p>
      Lorem ipsum dolor sit
      amet. Con minimim enami
      quis nostrud laboris
      nisi nonumi.
   </p>
   <div class="clearing">
   </div>
</div>
```



```
img
   {
    float: left;
    padding: 10px;
   }
```

```
<div class="bordered">
   <img src="flag.gif">
   <p>
      Lorem ipsum dolor sit
      amet. Con minimim enami
      quis nostrud laboris
      nisi nonumi.
   </p>
</div>
```



```
img
   {
    float: left;
    padding: 10px;
   }
.clearing
   {
    clear: both;
    height: 1px;
   }
```

```
<div class="bordered">
   <img src="flag.gif">
   <p>
      Lorem ipsum dolor sit
      amet. Con minimim enami
      quis nostrud laboris
      nisi nonumi.
   </p>
   <div class="clearing">
   </div>
</div>
```

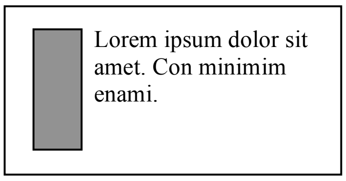


Figure 4: the 'drooping flag'. (top row) Without a float rule; (middle) same mark-up but with a float rule; (bottom) same mark-up but with an invisible clearing element.

What design manoeuvres have been prompted by the arrival of CSS? A number of special-purpose environments and tools are now available. The problem areas identified above were poor progressive evaluation; lack of abstractions over the CSS rules; hidden dependencies; and consistency and role-expressiveness. We take those in turn.

Progressive evaluation has been addressed by a variety of tools. Dreamweaver, already mentioned above as a WYSIWYG editor, was soon extended to show the effects of CSS rules. The browser Firefox [37], which is designed to incorporate third-party plug-ins, has acquired a 'Web Developer Tools' plug-in [38] that allows the CSS rules for a page to be edited from the browser. The special-purpose editor CSSEdit [39] displays the results of editing pages concurrently with the CSS being edited, as do many other tools.

Abstraction mechanisms over the CSS rule set have been provided by a few tools, among them CSSEdit, which introduces the notion of ‘groups’ of CSS rules. The groups are separated by comments in the CSS code, using a special syntax recognised by CSSEdit, so that they display in a hierarchy. (This is the same mechanism used by Dreamweaver, BBEdit etc to introduce templates as an abstraction mechanism at the HTML level). The group structure is chosen by the website developer: all rules associated with a certain area of the code, or all rules dealing with fonts, or whatever is thought useful. This mechanism does no more than localise the rules, but it considerably improves the visibility of CSS because related rules are now readily juxtaposed.

The hidden dependencies within CSS have been tackled by several systems, among them XyleScope [40], another specialised CSS editor. In Xylescope, one pane of the window renders the HTML page in the usual way; a second pane displays the CSS rules. Selecting any component of the HTML page brings up, in the third window, a view of the hierarchy of CSS rules that contribute to the selected component, so that it is possible to see whence it derives its properties. In the opposite direction, Xylescope can also locate and display all the HTML elements that match a given CSS selector.

The last problems identified were those of consistency and role expressiveness. For these, no tools have been developed, so far as the authors are aware, and indeed it is difficult to see how that could be done without changing the semantics of CSS. But that certainly does not mean that it is not an issue with real-world relevance. Far from it. Issues of consistency feature regularly in books on CSS (see [41] p. 142 ff., for the ‘drooping flag’ example) and they also feature regularly in exchanges on CSS discussion groups, where typically a less-experienced designer asks for help in making his web page display the way it was meant to, and a more-experienced designer explains the issue:

```
<snip header>
Dear All
I have created this page:
<http://www.tremonthouse.co.uk/room1.html> [At the time of this exchange, this
page showed a pair of “drooping flags”.]
The css is embedded. I cannot understand why the images don't extend the
footer. [I.e. why doesn’t the box go round them?]
Is this usual behaviour? I have included the images as content.
Is there a way around this please?
Thanks
Rich
----------------------
<snip header>
Yes, this is normal  behavior when floats (the images) are involved.
Add:
#footer {
     clear: both;
     background: #034400;
  }
....to make the footer 'clear' the floating images and come visible.
regards
     Georg
```

Figure 5: an exchange of messages on a CSS discussion list [42]

Exchanges such as this show clearly that the consistency issues that are exposed by a CDs analysis have real-world relevance, the more so since we imagine that only a minuscule proportion of puzzled website designers go so far as to pose their problem on a discussion group.

Very similar exchanges take place on the discussion lists enquiring why particular elements are present and how someone’s rule-set works, demonstrating the poor role-expressiveness that our analysis spotted.

3.3. CDs do relate to the real-world

The real-world relevance of CDs is fully established by these examples, in our view. For every likely difficulty predicted by a lightweight CDs analysis, we have observed subsequent developments – design manoeuvres – springing up to deal with it; the only exceptions being the consistency and role-expressiveness problems, which appear to be intractable, and therefore generate frustration in the users. And, as we argued at the start of this section, evidence of this sort has a validity that outweighs even the best laboratory study of user problems. The tools we have cited are commercial developments, without exception, and some of them are relatively expensive. They would not be marketable if there were not enough would-be designers who need them and are prepared to pay for them.

Unsurprisingly, the manoeuvres have illustrated the usual trade-offs. Alleviating the problems of HTML by adopting a template scheme entails grappling with the resulting premature commitment; using a separate notation as a means of 'decoupling' from HTML notation increases the cost of learning. Above all, using CSS to reduce the viscosity of HTML means accepting an extremely large increase in the abstraction level of HTML. CSS brings its own problems, and if tools are used to cope with them further trade-offs take place. Tools to improve CSS visibility, such as CSSEdit, impose a cost, because the CSS rules have to be carefully grouped—and at the same time they impose a degree of premature commitment. Tools to reduce hidden dependencies, such as XyleScope, increase the number of concepts to be understood, effectively increasing the abstraction level.

The story of HTML/CSS ends with a little moral. When we hear that mantra of "we need better tools", so common in development contexts, we might do well to infer that new abstractions are needed and quite possibly the notation is lacking in itself.

4. Towards 'methodologies'

The original conception of CDs [1] was as a practical tool for real-world use. Earlier sections of this paper have demonstrated the real-world relevance of the framework but not, as yet, its widespread take-up. The framework lacks a well-specified procedure or methodology, suitable for use by non-specialists. Looking around at other methodologies and their take-up or lack of it, suggests that potential users are encouraged by having a tool or a well-specified procedure; they can then feel more secure, more confident that they are doing at least the right kind of activity, and are less likely to overlook parts of the process. In addition, if the CDs can be well defined in the ontology of the tool, then the tool can support automatic recognition of possible usability issues related to CDs.

We shall present accounts of two efforts towards practical applicability. *Misfit analysis* arose from academia, and seeks to highlight misfits between the user's conceptual model and the conceptual model built into a device. Because it uses the same lightweight formalisation to represent both models, misfits show up readily. Our second example comes from the commercial world: this we shall call *the Microsoft procedure*. Each of these ventures has extended and developed the original formulation, though in very different ways.

4.1. Misfit Analysis

Blandford et al. [43] approached real-world usability by developing Concept-based Analysis of Surface and Structural Misfits (CASSM) and a specialised tool, the Cassata editor. CASSM is intended to be more structured and precise than CDs, while requiring less precision and effort than ERMIA. CDs are not a uniform set, but they all concern problematic aspects of the system design that might be regarded as poor fits between the system representation and the user's conceptualisation of it: viscosity represents the idea that something which is conceptually simple takes many device actions, for example.

This led to the notion of starting simply, by laying out the concepts the user is naturally working with and those represented within the system, and also those presented at the interface that mediate the interaction between user and underlying system. As discussed above, many of the CDs also relate to structural properties of the system, which can be expressed in terms of relationships between concepts – for example, that one concept *consists of* others, or that changing the value of one indirectly *affects* the value of another.

Over time, the ontology of CASSM was refined, through trial and error, to focus on those entities, attributes and relationships that matter to the user, are presented at the interface, and are included in the underlying system model. In addition, the actions needed to change the state of the system are considered in terms of whether they are classified as easy, hard, impossible or indirect (i.e. a value can only be changed by changing something else explicitly). This, in turn, has led to the distinction between 'surface' and 'structural' misfits.

Surface misfits are of two sorts. First, there may be concepts that are important to the user but that are simply not represented within the system, and for which the user has to find a work-around. For example, most central heating controllers do not incorporate the concept of a 'type of day' (e.g. working day or weekend), forcing the user to programme heating times for each day separately. Second, there may be system concepts that users are normally unaware of, and which hamper their work if they cannot learn about them easily. The 'float rule' and 'clearing element' discussed in section 3.2 are examples of such concepts.

These surface misfits are important, but are not well covered within the CDs vocabulary, apart from the notion of visibility. Since they do not refer to the underlying structure of the system representation, the CDs analysis has no place for them.

Structural misfits are usually exposed in a CASSM analysis via the relationships that have been defined between different entities and attributes, or via the constraints that have been implemented imposed by the system. Such misfits can make even conceptually simple tasks difficult or tedious to achieve with the existing system implementation. Examples of such misfits include viscosity, as discussed in section 3.1 (e.g. changing the appearance of all top-level headings) and hidden dependencies. CDs support reasoning about many such structural misfits.

CASSM analysis, then, has a fairly well-defined procedure, refined over analyses of several real-life systems. The basis is to take data from naturalistic use, to construct a descriptive model from that data, and then to reason from the model about surface and structural misfits. CASSM analysis is supported by a prototype data representation and analysis tool called Cassata, which includes routines to check for a subset of CDs which have been formally defined (downloadable with full instructions [44]).

Consider the HTML example presented in section 3:

- Web *pages* exist and have *links* to and from them;
- Web pages consist of a *head* and a *body*. In the type of site discussed here, they also include a *navigation bar* (and other elements that we do not enumerate here).
- The web *site* consists of pages.
- Every pair of *tags* has a *type* (e.g. paragraph or reference to another web page), *properties* (e.g. the font of the paragraph or the URL of the reference), and the *text* they surround.
- In basic HTML (without templates), the navigation bar is a concept that the user is *aware* of, but is *not explicitly represented* as such in the underlying system.
- The example above considers *paragraphs* of particular types (e.g. "heading 1" with a particular appearance).

These ideas are captured in the CASSM model below (Figure 6). As represented in this model, most concepts are singularly unproblematic, being present for user interface and system, and easy to create, delete, set or change. Nevertheless the CASSM model indicates that some of the concepts are by no means unproblematic. First, the 'linked-from' concept, the link that reaches one page from another page, known technically as the 'referrer' link, is absent from the interface and can only indirectly be set or changed (by changing links from the referring page). Next, the concept of a navigation bar is certainly absent from the underlying system, and indeed whether or not it is present at the interface is a moot point. Lastly the rendering of different types of paragraph, such as ‘h1’, is changed indirectly, by changing tag-pair properties.

	Concept (Entity / Attribute)	User	Interface	System	Create (entity) / Set (attribute)	Delete (entity) / Change (attribute)	Notes
E	web-page	present	present	present	easy	easy	
A	links-to	present	present	present	easy	easy	
A	linked-from	present	absent	absent	indirect	indirect	
E	head	difficult	present	present	easy	easy	
E	body	present	present	present	easy	easy	
E	tag-pair	present	present	present	easy	easy	
A	type	present	present	present	easy	easy	
A	properties	present	present	present	easy	easy	e.g. changing font size or pointing to location
A	boxed-content	present	present	present	easy	easy	
E	navigation-bar	present	difficult	absent	easy	easy	
A	links-to	present	present	present	easy	easy	
E	paragraph-type	present	notSure	notSure	easy	easy	
A	appearance	present	present	present	indirect	indirect	
E	site	present	present	present	easy	easy	

Figure 6: CASSM analysis of HTML code in a simple text-editor, Part 1: concepts

The relationships that matter in this case are those which define what an entity consists of and those which define how changing one attribute affects the value of another attribute. Most of these relationships are clearly represented, except the ‘referrer’ link of where pages are linked-to from (bottom row of Figure 7).

		Relationship		User	Interface	System
0	web-page	consists_of	head	present	present	present
1	web-page	consists_of	body	present	present	present
2	web-page	consists_of	navigation-bar	present	present	present
3	tag-pair.properties	affects	paragraph-type.appearance	present	notSure	notSure
4	paragraph-type	consists_of	tag-pair	present	present	present
5	site	consists_of	web-page	present	present	present
6	navigation-bar.links-to	affects	web-page.linked-from	present	present	present
7	web-page.links-to	affects	web-page.linked-from	present	absent	notSure

Figure 7: CASSM analysis of HTML code, Part 2: relationships

As discussed above, several CDs issues apply to a simple HTML editor:

- Viscosity: adding or deleting a page from the site involves repetitive changes to the navigation bar on every calling page; changing the style of a heading (for example) demands that the tags corresponding to every instance of that style be changed.
- Hidden dependencies: it is not possible to determine what pages link to the current one (and hence which need to be updated if the page changes); and
- Progressive evaluation and diffuseness are also issues.

Only some of these can be automatically identified by Cassata – namely, viscosity and hidden dependencies. These are reported by Cassata for this model as shown in Figure 8.

```
Repetition Viscosity Check
    "tag-pair.properties" affects "paragraph-type.appearance"
    "paragraph-type" consists_of "tag-pair"
POSSIBLE CASE OF REPETITION VISCOSITY:
to change "paragraph-type.appearance" user may have to change  all instances of
"tag-pair.properties"
====
    "navigation-bar.links-to" affects "web-page.linked-from"
    "web-page" consists_of "navigation-bar"
POSSIBLE CASE OF REPETITION VISCOSITY:
to change "web-page.linked-from" user may have to change  all instances of
"navigation-bar.links-to"
====
Hidden Dependencies Check
  "tag-pair.properties" affects "paragraph-type.appearance"
  "navigation-bar.links-to" affects "web-page.linked-from"
  "web-page.links-to" affects "web-page.linked-from"
POSSIBLE CASE OF HIDDEN DEPENDENCY:
there may be hidden dependency between "web-page.links-to" and "web-page.linked-
from"
```

Figure 8: output from automated CDs analysis using CASSM

The development of CASSM (and hence the Cassata prototype) has been highly exploratory. This applies both to the development of the ontology of the model, in terms of what features of the user's conceptualisation and the system representation to incorporate in the model, and to the definitions of CDs that have emerged through the work (and are now implemented in the Cassata tool). In contrast to the work of Roast et al on CiDa (section 5.1), our definitions of CDs have emerged from analysis of examples. Thus, the quality of the definitions is highly dependent on the quality and consistency of the examples from which they are derived. Three classes of CDs (2 variants of viscosity, 3 of premature commitment and one of hidden dependencies) have been tested thoroughly, and are therefore included in the Cassata test suite.

Just as the development of CASSM has been iterative and reflective, so the development of most models has also involved cycles of iteration. It would be nice to give a clear story that when we analysed a system, we uncovered CD problems that had gone unsuspected; but in practice the reasoning often goes the other way. For example, a system is believed to be viscous, so what has to be true of the model for that viscosity to be identified? This, in turn can be valuable in that it helps in the articulation of concepts and relationships that might otherwise go unrecognised. The recognition of these features could be central to the design of systems that better fit their users.

4.2. Personas: the Microsoft procedure

Whereas the CASSM methodology is based on a modelling formalism, the Microsoft procedure uses observation and discussion. Usability studies are used to gather feedback on APIs (Application Programmer Interfaces) throughout the API design process. In these studies, usability participants are given a series of tasks to perform using the API. These tasks typically require participants to perform the different types of activities described in the CDs framework. The CDs are then used to analyse observations made of the participants and to help improve the design of the API with respect to the particular set of tasks performed by the usability participants. In the Microsoft procedure the focus of discussion is on the tasks and how the participants performed those tasks. In particular, the procedure recognises different user personality types, unlike other reported applications of the framework. These types or 'personas' differ in their goals and methods, allowing a detailed appreciation of how users interact with the system's resources.

CDs showed their worth at Microsoft when they were used to analyse user difficulties with APIs for the .NET platform; many of the observations in the usability lab showed users struggling with the documentation for the APIs, and even after finding the code they ought to use, they continued to search for something.

Although the first response of the API implementation team was to rewrite the documentation, the usability team made a careful study of each dimension, and were persuaded by their analysis that the problem lay elsewhere.

> "Doing so made it clear to us that the reason that participants continued to search for other classes in the documentation was because the abstraction level of the classes that they stumbled upon in the documentation was too low. Participants expected to find classes that directly corresponded to the way that participants thought about the task. Instead, the classes they found were of a sufficiently low level of abstraction that participants could not relate these abstractions to the task they were working on." [46]

This initial success has led to much further use of CDs within Microsoft as an evaluation tool for APIs.

For evaluation purposes three main developer personas are distinguished – opportunistic, pragmatic, and systematic. These personas are well-developed and mature concepts, predating the adoption of the CDs framework. The Microsoft Visual Studio usability group began developing the personas around 2002. After three years they had amassed much data: approximately 90 usability studies with roughly 550 participants, approximately 150 individual sites visited, and approximately 20 different focus groups. They believed that this was a sufficient volume of data from which to identify patterns of developer behaviors and work styles. Furthermore, since the data had been collected from many different sources and in the context of many different studies they were confident in generalizing these patterns to scenarios that had not been the focus of any of the original studies.

To identify consistent behaviour patterns, they developed flowcharts describing how developers approach each of these tasks. The flowcharts described the different steps and goals for each developer on each task based on individual studies; eventually, these were distilled into the three personas mentioned, with the differences described in such a way that they could be understood and rationalized in terms of preferred steps and goals.

When the CDs framework was adopted, each persona was profiled in terms of the Cognitive Dimensions. So, for example, one specific work style of an opportunistic developer is to focus on the business problem that they are trying to solve, as opposed to the underlying technology that implements the business solution. This was captured by describing the abstraction level of an API that allows the developer to focus on the business problem instead of the underlying technology. More generally, for each of these three personas, it is possible to specify the 'developer profile', the value of each cognitive dimension that such a person will prefer, as described below. "The benefit of using the personas in this way is that they were already well established when we started using the cognitive dimensions. Being able to show how the cognitive dimensions related to the personas gave them additional validity within the company." **Error! Reference source not found.**

Also, the set of dimensions has been redefined, partly so that they can be specialised for their particular domain of application programming, partly to support the notion of persona. Some of the dimensions have been re-interpreted, such as 'abstraction level'; more importantly, new dimensions have been introduced, among them 'learning style', 'work-step unit' and 'penetrability'. These newcomers are of considerable interest.

(i) The *learning style* that an API supports is presented by Clarke [47] in terms that are specific to APIs but can readily be generalised to other information artefacts:

- "If each scenario requires a small number of classes and each of those classes has a small number of dependencies on other classes then the API supports an incremental and minimal learning style. *In other words, users only ever need to know about the parts of the API that are relevant to their goal and they can acquire this understanding bit by bit."* (The minimalist learning style has been well supported by research [49].)
- "If each scenario requires a large number of classes and each of those classes has a large number of dependencies on other classes then the API supports (or demands) a top-down or structured

learning style. *In other words, before being able to do anything useful with the API, the user needs to gain an understanding of the different components exposed by the API, the dependencies between different components exposed by the API, the architecture holding the API together and other conceptual information."* (Clarke, [47], our italics)

We can compare the difference between 'top-down' and 'incremental' learning styles to choosing between ready-prepared frozen meals versus cooking from individual ingredients. Each has its virtues. A ready-prepared meal can be warmed in the micro-wave, and the 'user' need know nothing about the individual ingredients. On the other hand, if it doesn't work well, not much can be done to rescue it or to improve it next time. Cooking from scratch demands much more background knowledge of what ingredients to use and how to prepare each one, but if it the result is unsatisfactory, the understanding of all the underlying ingredients allows small modifications for next time.

(ii) The *work-step unit* is likewise defined in terms particular to the API domain:

- "If the code that users have to write to accomplish a task using an API is completely contained within one local code block, and the code required can be written incrementally, the work step unit is defined as local incremental.
- If the code that users have to write to accomplish a task using an API is contained within multiple code blocks, or if the code requires the instantiation of multiple classes that interact, the work step unit is defined as parallel.
- If the code that users have to write to accomplish a task using an API is neither local or parallel but somewhere in between, the work step unit is defined as functional." [47]

It is somewhat harder to generalise this dimension to the complete class of information artefacts, writ large, although of course parallels do exist.

(iii) *Penetrability* "refers to the extent to which a developer must understand the underlying implementation details of an API and the extent to which the developer is able to understand those details. For example, an API that is basically a very thin wrapper over some lower level API probably requires that anyone using that API understands the low level implementation details before being able to use the API successfully." [47]

It can be seen from these three examples that to some extent the original conception of cognitive dimensions has been diluted; but on the other hand, the loss of purity is evidently accompanied by greater real-world applicability. The revised dimensions also work well with the pre-existing notion of a developer persona. When a system has been analysed on each CD, the system profile can be compared with each of the personas to create a developer profile. The usability team typically makes a presentation to an API design team explaining concepts of usability and introducing the CDs framework. "During the presentation, many people recognize the particular issues that are described as ones that they themselves have experienced. It becomes clear that the framework is useful for describing common issues that arise with APIs." Having gained the interest of the design team, some participants are recruited to write code using the API, and their observed behaviour is analysed and discussed with the aid of the CDs framework.

As a visual aid, a tool has been to developed to present the results of the analysis ([45]). The tool lists the 12 dimensions used at Microsoft, with the relevant evaluation questions for both the system being evaluated, *and* the developer persona being considered. For abstraction level, the questions posed are:

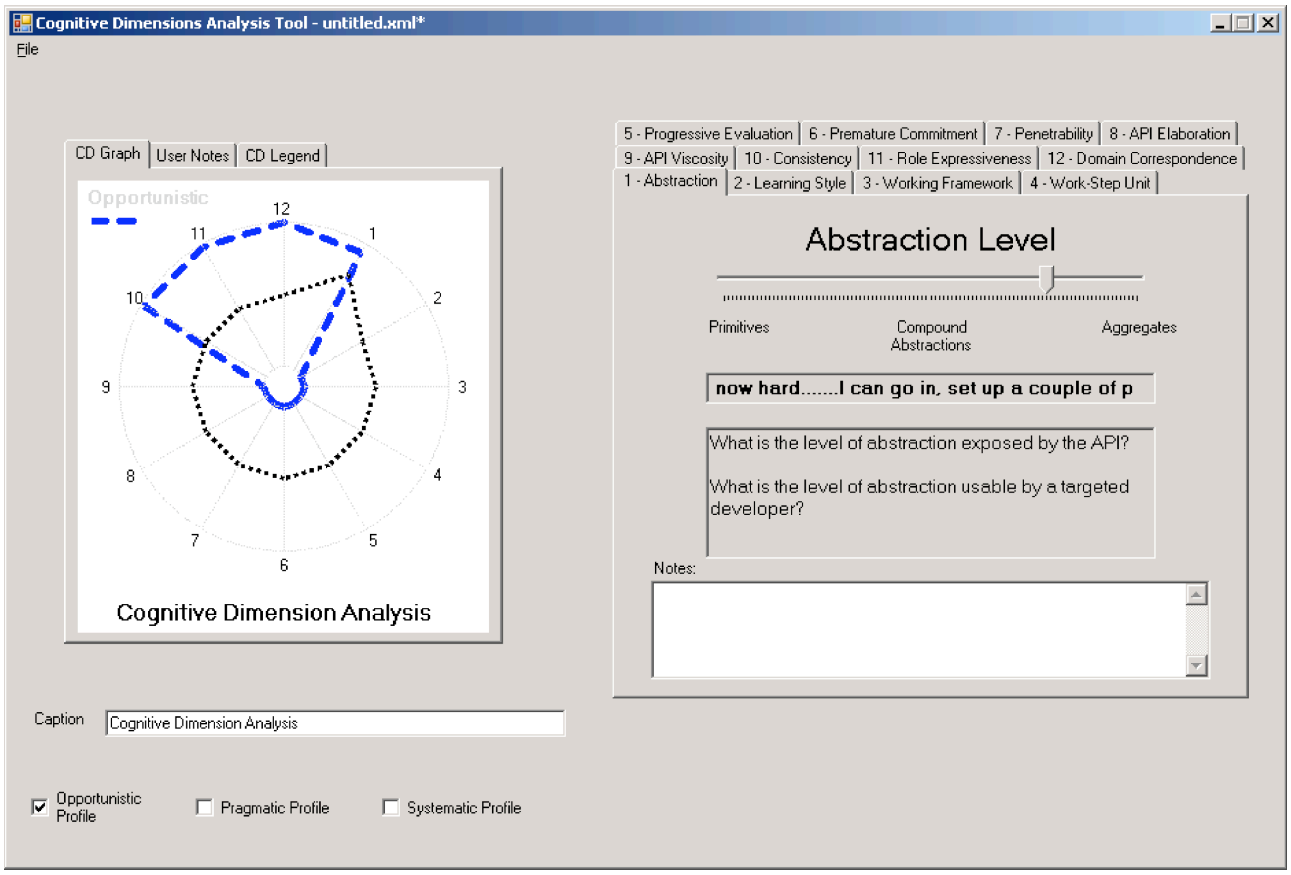
- What is the level of abstraction exposed by the API?
- What is the level of abstraction usable by a targeted developer?

Built into the tool are the characteristics of the three personas. In the present instance we are looking at an opportunistic developer, whose preferences are shown by the blue line on the radar chart, Figure 9:

"[The example] shows that this persona prefers APIs that have small work-step units (the inner edge on the work-step unit scale [spoke 4]), support progressive evaluation at the line of code level (the inner edge on the progressive evaluation scale [spoke 5]), and have rich role expressiveness (the outer edge on the role expressiveness scale [spoke 11]). " [45]

The black line on the chart shows the evaluation of a particular API. The evaluation shown is for fictional data, an evaluation of a purely hypothetical system, but it indicates quite clearly that that system is a very poor match for the characteristics of the targeted developer persona. Clarke reports that "We provide the graphs to [design] teams as a means to visualize how well their API meets their customer's requirements. We find that people really react quite strongly to this visual representation of the API study results." **Error! Reference source not found.**

In the Microsoft procedure, the notion of 'activity type' plays no part: this is an effective simplification, since the participants are all observed doing the same activity and there is no intention to generalise beyond that activity, and the notion of persona – not included in the 'classic' CDs framework – fulfils many of the purposes of the distinction between activities. It will be interesting to see whether one of these notions



Abstraction Level

now hard.......I can go in, set up a couple of p

What is the level of abstraction exposed by the API?

What is the level of abstraction usable by a targeted developer?

Caption: Cognitive Dimension Analysis

[✓ Opportunistic Profile] Pragmatic Profile Systematic Profile

proves superior to the other, or whether both are ultimately needed.

Figure 9: radar diagram comparing the fit of a fictional system (black line) to a developer persona (blue line) for each CD, numbered 1-12 **Error! Reference source not found.**

Microsoft's work with the Cognitive Dimensions framework has been summarised by Clarke [45]:

"At the very least, the cognitive dimensions framework provides developers with a common vocabulary with which to talk about and discuss API usability. The benefits of such a common vocabulary are twofold:

- First, the language shapes the distinctions that are considered important to attend to. By explicitly describing each of the dimensions, it is more likely that they will be attended to.
- Secondly, the language lets developers discuss issues using terminology that they can assume will be understood by others who are also familiar with the framework.“ [45]

Foregrounding the concepts of shared terminology and explicit description bears a striking resemblance to Green’s original vision, cited in the introduction, of CDs as ‘discourse tools’ [2].

4.3. The future of methodologies

Of the two methodologies described above, CASSM’s misfit analysis requires a modeller who either understands the system to be analysed or else has access to experienced users; ideally, the modeller must have enough empathy with users to be able to spot certain types of difficulty independently. The output is a model of the system in a lightweight formalisation, although those who used CASSM have suggested informally that some part of its contribution comes from the understanding that is achieved by the process of building the model, as well as from the model itself. The Microsoft procedure uses observational data, worked through and analysed by a team, and its output is a cognitive dimensions analysis differentiated by the user personas.

Both methodologies are limited by requiring a working prototype. (In principle they could manage without, but they are much easier with one.) An interesting question arises when what is to be evaluated is a proposed modification to an existing, commercially-available system of some complexity; making a prototype incorporating the modification then becomes prohibitively expensive. Blackwell et al [50] invented a third methodology to cope with precisely that circumstance, colourfully named Champagne Prototyping. Their scheme relied on access to a small number of very experienced users, who were asked to discuss pros and cons of the modification as exhibited by stationary screens-shots. Their answers were then analysed using the cognitive dimensions and attention investment ([11]) frameworks. Champagne prototyping (so called because their users were paid with a bottle) avoided the need to build a working prototype, because the screenshots were simply edited versions of the screens to be seen in the unmodified original application.

Any type of methodology is subject to the economic rules of costs and benefits. If the effort invested fails to yield an effective return, that methodology will see little use. It will be interesting to see whether any of these three approaches becomes widely accepted.

5. Redefinition and refinement

It is clear that the cognitive dimensions are birds of more than one feather. They are meant to be reasonably orthogonal and reasonably precise, to have about the same level of generality and to be drawn from the same universe of discourse, but these lofty intentions were not completely realized. Much of the difficulty comes from the vagueness with which they have been expressed. Vagueness certainly helps to make them accessible to new users but also helps to foster misunderstandings and even allows reshaping of the dimensions, problems noted already in the brief historical account given in Section 1. Vagueness also breeds inconsistencies and overlaps, so that in different publications the ‘same’ dimension is treated differently, or the same notational feature is given different labels.

Can the dimensions be better defined and where necessary be given an improved specification, to avoid overlaps, or can a better set of dimensions be developed?

In this section we consider two approaches; both are ongoing work. The first approach, CiDa, attempts to formalise the existing dimensions, while yielding useful warnings of overlaps and imprecisions. The second approach, ‘refactoring’, although currently very speculative, attempts to separate the dimensions into different groupings relying on different foundations. (Note also that the ‘CASSM’ technique, presented above, leads to some refinements of existing dimensions.)

5.1. CiDa

Formal definitions of dimensions are not easy to create, and the implications of any particular proposed formal definition are not always immediately apparent. CiDa [27],**Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.**,**Error! Reference source not found.** is a tool for experimenting with definitions.

The primary issue for any proposed definition is whether it is *valid* with respect to what it is intended to define. In the case of cognitive dimensions, the framework in which satisfactory definitions could be posited is itself unclear. It has already been argued earlier in this paper (Section 2) that a definition must be developed with respect to cognitive behaviour and structural system properties, and possibly specific domain knowledge as well.

The CiDa tool uses a state-based representation of a system and a formal definition of possible user goals and actions. Given this setting CiDa exhaustively checks the behaviours of the modelled system in order to identify invariant features of them that are sufficient to fulfil general dimension definitions. The output of this analysis is a list of the possible dimension examples found for the target system. The list of examples can be employed both as the basis for exploring the validity and also as an explicit basis for exploring how the dimension examples vary between similar target systems **Error! Reference source not found.**.

CiDa has been developed as a tool for theory development, operating by simulating usage of a system. The simulated system is represented as a state machine in terms of all possible states of the system (the "state space"), all the actions possible in each state and the state each action leads to. Possible user goals are represented as "sub-regions" within the state space. Thus if we consider a goal, such as, "no more pending emails", the CiDa view of it is the set of all system states in which the pending emails list is empty. Finally, CiDa also needs a definition of specific cognitive dimensions. Each definition has to be expressed in terms of patterns of possible of goals (and their combination) that may be fulfilled and in what order actions in the simulated system allow goals to be fulfilled. Thus for any one proposed definition of a cognitive dimension, CiDa is able to report on each dimension instance it finds in the simulated system that matches the definition. A dimension instance is the dimension concept instantiated to specific goals and actions.

To describe the operation of CiDa, we shall consider the notion of knock-on viscosity. Green [3] distinguishes two types of viscosity: ‘repetition’ viscosity, in which a change to an information structure requires many individual actions (such as changing all the headings of a paper to bold face); and ‘knock-on’ (or ‘domino effect’) viscosity, in which a single change creates inconsistencies which have to be subsequently corrected by the user. A common example of knock-on viscosity is adding a new section early on in a paper with numbered sections; this process will require all subsequent sections and section references to be re-numbered, unless the user is working with a smart editor supplying useful abstractions.

We shall use that example as our working context. There are at least two goals in our example: (a) to add a new section, and (b) to have all sections properly numbered, as well as the action X that creates the new section. The user’s difficulty occurs when (b) has been met and now (a) has to be met as well, because adding a new section may disrupt the existing ordering.

CiDa starts at the start state, and traverses the entire state space applying each possible action in each possible sequence. In the case of testing for the occurrence of knock-on viscosity just described, the operation of CiDa is as follows:

```
Consider each state reached by some action sequence
    If, in that state, goal (b) is met but goal (a) is not:
        Examine the effect of applying the action X in that state.
              If X achieves goal (a) and negates goal (b), then it counts as
              an instance of knock-on viscosity involving a, b, and X.
```

An instance of a cognitive dimension is an example of a general dimension identified by the goals and actions that characterise the example, such as primary goal, secondary goal and user-action. For CiDa's definition of cognitive dimensions, the goals and actions that determine an instance of a dimension are formally defined parameters within its definition.

When CiDa is employed to perform a generic analysis of a simulated system every possible instance of each dimension is checked and the resulting list of properties output. Although the number of instances resulting from such an analysis can be considerable it is of significant value when exploring the dimensions, since it is not subject to biases of the intended domain and can support consistent comparative examination of alternative designs. In a related study the "extent" of a dimension within a specific system has been characterised in terms of the number positive instances of the dimension generated by CiDa's analysis of the system **Error! Reference source not found.**.

CiDa analysis has been applied to viscosity, premature commitment and secondary notation, yielding unexpected insights in each case. For example if we consider *modifying* a notation, knock-on viscosity and repetition viscosity can be shown to change as a result of the modification but in complementary ways. Imagine a tool that provided the seamless combination of two notations with the same purpose, say a visual programming language and a textual programming language. The tool could be seen as offering a hybrid facility by which its users could freely access a program in visual or textual form. Proposed formal definitions of dimensions allow for the assessment of the tool to be characterised in terms of the notations it combines, and yield the following results:

- the repetition viscosity of the combined notation is the same as that of the notation with the *lowest* viscosity;
- the knock-on viscosity is the *sum* of those of the notations.

We can conclude that combining languages ensures that program modification can be simpler to perform, but it also increases the potential complexity of modifications if the simplest method is not used **Error! Reference source not found.**.

Although formalisation using CiDa has worked quite well for the dimensions mentioned, viscosity, premature commitment and secondary notation, it is not so easy to see how to apply it to dimensions that depend more obviously on characteristics of the human cognitive system. Evidently the dimensions are birds of more than one feather. Rather than try to disguise their differences, in the next section we explore the possibility of separating them—and, where necessary, redefining them—according to their foundations.

5.2. Refactoring the Cognitive Dimensions

If we are to achieve satisfactory precise (possibly even formal) definitions of the cognitive dimensions, we need to understand their foundations. In the present list, there are dimensions that are almost entirely cognitive in content. For example, discriminability depends on the properties of the parsing system.

On the other hand, some of the dimensions have very little cognitive content. But there is a further division to be noted. Some are properties of the notational structure, possibly alleviated by the environment; others seem to be imposed by the environment in which the notation is used. Hidden dependencies, for instance, seems to be purely structural; the dependencies within spreadsheets, for example, are always expressed as one-way dependencies (cell A draws data from cell B), and that is a property of the information structure/notation, possibly alleviated by auditing tools that can reveal the inverse dependencies. Premature commitment, in contrast, is imposed by environments that constrain the order of actions against the flow of the dependencies in the notation.

Should we, then, divide the dimensions into groups? Would that simplify things? Unfortunately we suspect that with the current list and their definitions, the division is not clean. Viscosity can be construed as a mismatch between the user's idea of a single action and the ideas built into the device. Is that structural or cognitive?

So the problem is to consider how we can refactor the whole set in such a way that structural and cognitive aspects are differentiated and made explicit, and where possible can be addressed separately: where 'refactoring' is…

> "… the process of rewriting written material to improve its readability or structure, with the explicit purpose of keeping its meaning or behaviour" – Wikipedia **Error! Reference source not found.**

It would, of course, be important not to lose the simplicity and 'value for time' approach of the CDs; in fact, if those were to get lost in the process, much of the *raison d'être* of the framework would have been lost. But a successful refactoring of CDs into RCDs (Refactored Cognitive Dimensions) would confer advantages ranging from the aesthetic to the practical.

In this section we report on preliminary investigations in this area. Although a full analysis on these lines remains to be done, we report it in the manner of a prolegomenon to any future formalisation: attempting to formalise dimensions from differing universes as though they were all from the one universe is likely to fail.

How can we go about this refactoring? A clue can be gained from a thought-experiment in applying a CDs analysis to a system that is self-evidently non-cognitive: the evolution of genetic algorithms. (This Darwinian-inspired search process starts with a set of candidate solutions to a problem, defined by features known as 'chromosomes', then picks two good candidates, and combines their features with possible mutations to create a set of offspring, the entire process then restarting with a new generation.) Whilst the detailed analysis is beyond the scope of this paper, some of the cognitive dimensions seem to apply, while others have no apparent applicability. Examples of relevant dimensions are:

- Hidden Dependencies litter the design of the genome representation and have significant effects on whether an evolution will be successful or not;
- Some degree of viscosity is necessary to prevent the evolutionary mechanism running away;
- Work-step (amount of change per cycle) is important for the performance of the algorithm. Search spaces with many severe local minima may require a high work-step to make progress.

However as this is not a cognitive process, analyzing these from a cognitive perspective should make little sense; something more fundamental must be at work here. What has happened is that the CDs analysis has revealed *information structure issues* to do with the exploration of the search space. Only those dimensions that are search-based have contributed to the analysis. This is what would be expected, given that genetic algorithms are a search mechanism, not a cognitive mechanism.

There are other fields where the distinctions between truly-cognitive and information structure dependencies are even harder to untangle. For example, computer security systems typically involve complex user interfaces, frequently designed for security experts, which interact with complex, massively interdependent data structures. It seems likely that a more formal separation between the information structures and the interface is essential if progress is going to be made on the usability of such systems.

Let us now consider the problem of how to go about such a refactoring.

1. The actual user interface can only contain cognitive assistance, so it can be abstracted away from without loss of generality. Hence consider an abstract representation of the interface onto the information artefact.
2. Consider an aim of a user and qualitatively evaluate how difficult it would be to write a search algorithm to achieve that aim through interactions with the abstract interface. Record all the difficulties that would be involved in doing so; these represent the difficulties associated with the information structure, and should be removed from further consideration as cognitive properties.

3. Other difficulties are not related to the interaction with the information structure and so can be considered to be cognitive issues.

Notice that item 2 is deliberately vague: the idea is not to actually construct the search procedure, just to specify it, in order to understand what aspects of the system makes it difficult. In particular, no specific search procedure is defined. Any and all may be considered for use.

For example, consider a system that displays a ‘Confirm/Abort’ dialog box after every operation. This feature has essentially no effect on the search procedure, it performs the operation and then accepts ‘Confirm’. So the feature does not represent an information structure problem, but it may be a cognitive problem (viscosity) dependent on the context.

The basis of this refactoring is that people use search techniques extensively when interacting with information artefacts. They may be searching through the state-space trying to find the correct state for their goal (see Thimbleby **Error! Reference source not found.** for an analysis) or they may be searching for the available actions from their current state, e.g. by poking through a menu system **Error! Reference source not found.**. The search-based issues are the fundamental, non-cognitive issues. For example, some state spaces contain areas that can only be reached by one or two paths (known in graph theory as bridges or cut-edges). A search procedure that gets into such an area may have difficulty in returning to where it started, and we have all experienced that kind of frustration using a new system. Still worse, the information structure may have ‘trap-doors’, one-way sections creating dead ends or areas that can be left only with great difficulty.

Those usability issues that are revealed by considering the difficulties of searching the state space can be regarded as non-cognitive. The remaining issues are likely to be cognitive.

So for example discriminability has no effect on a typical computer program, which uses canonical internal representations, hence such an issue would not be detected as a structural issue, and must be considered a cognitive issue.

Limitations

There are limitations to this analysis.

First, if the search algorithm cannot be developed the analysis fails (e.g. the game of Go). This limitation is probably not too serious, given that CDs are generally used to understand, communicate and manipulate properties of systems that can be built as opposed to as a tool for understanding AI research problems. Its main effect is diminishing the value of RCDs for the analysis of complex thought experiments.

Second, since the original CDs framework is not exhaustive, neither will be the RCDs framework. Incompleteness is a trade-off that was made when CDs were originally developed--for example, they deliberately exclude the visual rendering of an interface. RCDs take this approach even further by the analysis of an abstract representation of the interface, not the interface itself. This is probably more significant and makes the consideration of the search procedure more difficult; but it is unlikely to be fatal to the analysis of most practical problems.

Example: refactoring viscosity

An RCDs analysis of the properties of viscosity shows that it can be roughly divided into *cognitive* and *structural viscosity*. *Cognitive* viscosity is typically caused by a misfit between a user interface design and the user’s concept of a single operation. For example, the previously discussed hypothetical interface that requests confirm/abort clearance for every operation potentially has a viscosity problem. A single operation from the user’s perspective has become two or more operations from the system’s perspective. As the search algorithm for this interface is trivial, the viscosity is a cognitive, not a structural issue.

Similarly *structural* viscosity is caused by an information structure that has a misfit with the user’s perspective. For example, if the user of a system considers items in it to be a list, they may believe that inserting a new item should be a single task. However in some cases, the insertion involves many updates to other locations. A classic example of this problem is the numbered reference system for citations (as in this Journal), which can cause frustration and wasted time for authors lacking expensive citation-handling software. In the original CDs formulation such a misfit would be an example of hidden dependencies. Without appropriate support from the system hidden dependencies will result in high structural viscosity, as a single operation from the user's perspective requires a large number of operations to maintain the data-structure.

The RCDs analysis clearly separates these two types of viscosity that were previously conflated. With this perspective, the solution is clear; the data structure needs updating to be list-like rather than array-like, to re-align the data structure's perception of a 'single' operation with the user's perception. This will result in a significant decrease in the viscosity, through a change in the structure of the hidden dependencies.

A suitable refactoring of the CDs would allow the consideration of such design manoeuvres with a substantially increased understanding of their effects on either the cognitive or information/structural aspects of the system. It would also simplify the consideration of how changes to the underlying information structure can affect the cognitive aspects. And finally, it would smooth the path to satisfactory formalisations and definitions of the dimensions, with the attendant benefits of stability and comprehensibility.

6. What areas remain dark corners?

Although the paths followed by the CDs framework have been varied and unexpected, there remain areas that have received almost no attention, yet are of increasing relevance to contemporary interaction design. Each of the three examples listed below has teased at least one of the authors, while trying to find a clear analysis of the pros and cons of some instance of interaction design.

6.1. Media types

The properties of different media have not been properly explored -- hardly even at the hand-waving level. In so far as they have received any mention at all, media have been divided into *persistent*, such as paper, and *transient*, such as speech. Transient media place strong demands on the notation. The paradigm example is creating a program: if spoken, dictation is likely to be from start to finish, which means that declarations must be made early; but if written, declarations can be made last, when it is known what variables are needed. This leads to the conjecture that, in general, notations with many internal dependencies may be difficult to use in speech-based system because the transience is likely to impose order constraints, leading to premature commitment.

But the distinction between persistent and transient media is too absolute. Media vary in how much history they reveal. If you cross out a word on paper, it leaves a scar; if you delete a number in a spreadsheet, it leaves a gap, which might be seen as a scar. But if you delete a word from a document in a word-processor the neighbouring words will fill the gap, so word-processors are 'self-healing' whereas paper, and to a lesser extent spreadsheets, are 'cicatrizing' (i.e. they form scar tissue). Whether that is important depends on whether the scar is useful to a person working with the document. Sometimes it can act as a reminder and thus reduce memory load, a phenomenon observed in a study of diary usage **Error! Reference source not found.**.

Taking the concept of a 'scar' a step further brings us to systems with history lists and history managers. Unix is no doubt the star example, with a sophisticated tool allowing users to repeat commands with alterations. Experienced users clearly find it very useful.

At present we have little idea how to treat any of these issues, yet they are probably important components of usability.

6.2. Temporal aspects

One dark corner leads to another. The only notion of time-constraint in the framework as originally proposed was that some environments constrained order of actions, producing possible premature commitment. But time constraints are much more varied, and can sometimes be cleverly used by interaction designers: for example, cash machines (automated tellers) make you recover your credit card before they give you any money, which prevents the easy 'post-completion error' of collecting the money - which is the primary goal - and forgetting to take the card. (Designers of certain railway ticket machines have yet to rediscover this trick, and it seems that credit cards are regularly left in such machines at Leeds station. An anonymous referee adds that "Not all ATM machines are designed to avoid the post-completion error. This would seem to be a particular issue in USA airports.")

In other cases, actions have to be performed in a set order. If the user cannot determine the necessary order, the system suffers from premature commitment. Moreover, we know from studies of typing that order errors are frequent and in some cases highly consistent - for example, in 80% of transposition errors (errors in which neighbouring letters are transposed) the hands are transposed as well, while sequences requiring a doubled letter or number are frequently typed with the preceding character doubled instead **Error! Reference source not found. Error! Reference source not found.**. Thus, the imposition of a set order can sometimes risk increasing the error-proneness.

How can users ‘change the order’ of actions, or rather, how can they take further actions to say ‘please treat earlier actions as though performed in a different order’? The simplest way is for them to erase the most recent action, then the one before, etc (like using a Delete key in typing), until all errant actions have been erased; then to recreate them in the correct order. This is a very viscous solution, but in certain syntax-directed editors it is the only option available. The viscosity can be reduced by a history mechanism, as just discussed, but here we note the usual trade-off between reducing viscosity and increasing abstraction level.

It is quite possible that the analysis of the properties of media and the analysis of temporal constraints can be brought together into a single extension of the CDs framework.

6.3. Collaborating representations

In section 3.1 we noted that the prolixity of HTML code could be reduced by using ‘structured text’, a less diffuse representation of the same structure, and observed that this is an example of ‘decoupling’, a form of work-around manoeuvre [6]. We shall refer to sets of representations offering complementary advantages as ‘collaborative’, especially when the representations are all capable of being edited, each propagating the changes to the others. The fields of information visualization and instructional research have generated many examples of multiple representations, and shrewd analyses have been made [52], but both their examples and their analyses are usually of the single activity of comprehending or reasoning from the information presented. (See Blackwell and Engelhardt [60] : “In our survey, it is obvious that the majority of the taxonomies developed so far in diagram research concern representation-related aspects.”)

When other types of activity are taken into account, in particular activities requiring manipulation or transcription, particular representations or media are likely to have their own advantages. A familiar example is the use of pencil and paper for sketching before commitment to a more rigid representation. In this instance, it is easy to see the advantages of ‘decoupling’: pencil is erasable and therefore has less viscosity than ink or carving; paper is a random-access medium and therefore imposes less premature commitment. Among the support tools for HTML/CSS mentioned in section 3, we have already cited structured text as an example of collaborating representations; further examples mentioned there include the increase in visibility given by CSSEdit, which localises CSS rules in a hierarchy, and by Xylescope, which exposes hidden dependencies in CSS. Both of these allow operations on each representation, so that the user can choose which is convenient—unlike, for example, spreadsheet auditing tools that draw arrows to trace precedents or dependents but do not allow any operations on that representation. At a much higher level of complexity, integrated development environments (IDEs) for programming frequently employ collaborating representations. An example at the instructional level is BlueJ [61] which presents both a graphic representation of a Java program and a code representation, either of which can be edited.

All of these examples, so far as the authors are aware, have been constructed from intuition aided by trial and error. We suggest that a better understanding of how representations can collaborate would be timely and useful, and that a CDs analysis could play an effective role.

7. Conclusions

The intention of this paper was to consider how far the Cognitive Dimensions framework has met the original expectations and where it might be heading. We have revisited the theoretical basis behind the CDs, and restated the original aim of seeking an evaluation or comparison technique that was invariant to changes in the presentation of a device or to changes in the domain of application—i.e. a technique that gave the same answers even if a device was 'reskinned' or was applied to a different domain.

It remains unclear whether any demonstrably complete list can be drawn up. We incline to believe that it cannot; the more so as the original vision is under pressure to include dimensions with a knowledge component, driven by the desire for real-world applicability. A good example of such a candidate is the concept of 'penetrability', supporting effective operational models, proposed as a possible addition to the list of dimensions; we showed that this dimension includes a knowledge component, and is therefore outside the original boundaries—but maybe those boundaries were set too narrow.

Next, we have demonstrated that the CDs framework has real-world relevance. This was shown by what we believe is a novel technique: we analysed a system (HTML/CSS) to predict the user difficulties, and then showed that specialised tools have been developed and marketed to deal with every one of those difficulties—except the apparently intractable difficulties of inconsistency: and that those remaining, festering difficulties were one of the subjects of requests for help on a CSS discussion list.

At the same time, however, we noted problems. There are occasional problems of comprehension because the dimensions are so vague; the lack of well-defined procedure disturbs some would-be users. Such problems may be reduced by developing well-defined methodologies. We have reported two approaches. One approach is based on an extension from the original CD framework to misfit analysis, more specifically Concept-based Analysis of Surface and Structural Misfits. CASSM offers a slightly tighter formalism than CDs, and allows a clear procedure to be followed, supported by a specialised tool which both records the analysis and offers a degree of automated detection of usability problems. A second approach, now in regular use at Microsoft, defines a clear procedure--and cheerfully changes the ground rules in order to get an effective grip on local problems. It is noteworthy that in seeking real-world application, both these ventures have extended the original formulation of the framework, although in very different ways.

But there are problems with the existing list of dimensions: they are vague, and they overlap. We showed that improvements are possible and are being explored. Better precision can be attained piecemeal by state-based analysis such as that deployed by the CiDa tool; and the problems of overlap appear to be soluble, at least in principle, by separating out the search components from the cognitive components to give refactored Cognitive Dimensions, RCDs, which offer some new insights. We have also noted that some areas of interaction design remain unexplored as yet, selecting temporal issues, properties of media issues, and the analysis of 'collaborative representations' as examples.

There may be outstanding dark corners, but CDs have met their original expectations in delivering useful insights -- particularly about programming notations -- and have been taken up and used in practice, indicating substantive usability and utility. The price that is paid for take-up is that they have taken their own course, wayward but illuminating.

8. References

All URLs for web-based references were accessed 13 March 2006.

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

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

[3] Green, T.R.G. (1990) The cognitive dimension of viscosity - a sticky problem for HCI. In D. Diaper and B. Shackel (Eds.) *INTERACT '90*. Elsevier.

[4] 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.

[5] Green, T.R.G., Petre, M. and Bellamy, R. K. E. (1991) Comprehensibility of visual and textual programs: a test of Superlativism against the 'match-mismatch' conjecture. In J. Koenemann-Belliveau, T. Moher, and S. Robertson (Eds.), *Empirical Studies of Programmers: Fourth Workshop*. Norwood, NJ: Ablex. Pp. 121-146.

[6] Green, T.R.G. and Blackwell, A.F. (1998). Design for usability using Cognitive Dimensions. Tutorial session at British Computer Society conference on Human Computer Interaction HCI'98. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf

[7] Green, T.R.G. (1991) Describing information artefacts with cognitive dimensions and structure maps. In D. Diaper and N. V. Hammond (Eds.) *Proceedings of "HCI'91: Usability Now", Annual Conference of BCS Human-Computer Interaction Group*. Cambridge University Press.

[8] Petre, M. and Green, T. R. G. (1992) Requirements of graphical notations for professional users: electronics CAD systems as a case study. *Le Travail Humain*, 55(1), 47-70

[9] 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

[10] Benyon, D., Green, T.R.G. and Bental, D. (1999) *Conceptual Modelling for User Interface Development*. London and Berlin: Springer (Practitioner Series).

[11] Blackwell, A.F. & Green, T.R.G. (1999). Investment of attention as an analytic approach to Cognitive Dimensions. In T. Green, R. Abdullah & P. Brna (Eds.) *Collected Papers of the 11th Annual Workshop of the Psychology of Programming Interest Group* (PPIG-11), pp. 24-35. http://www.cl.cam.ac.uk/users/afb21/publications/PPIG99.html

[12] Kadoda, G.F. (1999) Desirable features of educational theorem provers - a cognitive dimensions viewpoint. *Proceedings of the 11th Psychology of Programming Interest Group Annual Workshop* (PPIG).

[13] Kadoda, G. (2000) A Cognitive Dimensions view of the differences between designers and users of theorem proving assistants. In A.F. Blackwell & E. Bilotta (Eds.) *Proceedings of the Twelfth Annual Workshop of the Psychology of Programming Interest Group* (PPIG). http://ppig.org/papers/12th-kadoda.pdf

[14] Blackwell, A.F. and Green, T.R.G. (2000) A Cognitive Dimensions questionnaire optimised for users. In A. F. Blackwell and E. Bilotta (Eds.) *Proceedings of the 12th Annual Meeting of the Psychology of Programming Interest Group* (PPIG-12). Cosenza, Italy: Editoriale Bios s.a.s., pp 137-152. For the questionnaire itself: http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf

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

[16] Ball, L.J. & Ormerod, T.C. (1995). Structured and opportunistic processes in design: A critical discussion. *International Journal of Human-Computer Studies*, 43, 131-151

[17] Olson, J.R. and Olson, G.M. (1990) The growth of cognitive modelling in human-computer interaction since GOMS. *Human-Computer Interaction*, 5, 221-265.

[18] Nielsen, J., and Molich, R. (1990). Heuristic evaluation of user interfaces. *Proc. ACM CHI (Computer-Human Interaction) '90 Conf*. (Seattle, WA, 1-5 April), 249-256. See also: http://www.useit.com/papers/heuristic/heuristic_evaluation.html

[19] MP3 skins: www.mp3daze.com/skinskjofl.htm

[20] Johnson, P., Johnson, H., and Hamilton, F. (2000) Getting the knowledge into HCI: theoretical and practical aspects of task knowledge structures. In Chipman, S., Shalin, V., and Schraagen, J. (Eds.) *Cognitive Task Analysis*. Mahwah, NJ: Lawrence Erlbaum and Associates. pp. 201-214.

[21] knight and knave puzzles: http://philosophy.wisc.edu/lang/211/puzzles.htm

[22] Kimura, T. D., Choi, J. W. and Mack, J. M. (1990) Show and Tell: a visual programming language. In E. P. Glinert (Ed.), *Visual Programming Environments: Paradigms and Systems*. IEEE Press.

[23] The C Book (an online tutorial): http://publications.gbdirect.co.uk/c_book/chapter5/pointers.html

[24] Payne, S.J., Sime, M.E. and Green, T.R.G. (1984) Perceptual structure cueing in a simple command language. *International Journal of Man-Machine Studies*, 21, 19-29

[25] Gilmore, D.J. and Green, T. R. G. (1988) Programming plans and programming experience. *Quarterly Journal of Experimental Psychology* 40A(3) 423-442.

[26] Green, T.R.G. and Borning, A. (1990) The Generalised Unification Parser: modelling the parsing of notations. In D. Diaper and B. Shackel (Eds.) *INTERACT '90*. Elsevier.

[27] Roast, C.R. (1997) Formally comparing and informing design notations. In H. Thimbleby and B. O'Conaill and P. Thomas (Eds.) *People and Computers XII*, 315-336. Springer-Verlag.

[28] Roast, C.R. (1988) Designing for delay in interactive information retrieval. *Interacting with Computers*, 10 (1) 87-104

[29] Blackwell, A.F., Britton, C., Cox, A. Green, T.R.G., Gurr, C.A., Kadoda, G.F., Kutar, M., Loomes, M., Nehaniv, C.L., Petre, M., Roast, C., Roes, C., Wong, A. and Young, R.M. (2001). Cognitive dimensions of notations: design tools for cognitive technology. In M. Beynon, C.L. Nehaniv, and K. Dautenhahn (Eds.) *Cognitive Technology 2001* (LNAI 2117). Springer-Verlag, pp. 325-341.

[30] Cox, A.L. & Young, R.M. (2000). Device-oriented and task-oriented exploratory learning of interactive devices. In N. Taatgen & J. Aasman (eds*.), Proceedings of the Third International Conference on Cognitive Modeling* (pp. 70-77). Veenendaal, The Netherlands: Universal Press.

[31] Payne, S.J. (1991) A descriptive study of mental models. *Behaviour and Information Technology,* 10 (1), 3-21.

[32] Pratchett, T. (1986, 1994) *The Light Fantastic.* London: Corgi (paperback edition).

[33] Gray, W.D., & Salzman, M.C. (1998). Damaged merchandise? A review of experiments that compare usability evaluation methods. *Human-Computer Interaction*, 13(3), 203-261

[34] Dreamweaver: http://www.macromedia.com/go/gnavtray_dwmx_home

[35] BBEdit: http://www.barebones.com/products/bbedit/index.shtml

[36] Lie, H.W. and Bos, B. (1999) *Cascading Style Sheets: Designing for the Web*, second edition. Harlow, England: Addison-Wesley (Pearson Education imprint).

[37] Firefox: http://www.mozilla.org/products/firefox/

[38] Developer Tools: http://chrispederick.com/work/webdeveloper/

[39] CSSedit: http://www.macrabbit.com/cssedit/

[40] Xylescope: http://www.culturedcode.com/xyle/

[41] Lowery, J.W. (2005) *CSS Hacks and Filters*. Indianapolis, Indiana: Wiley.

[42] css-d@lists.css-discuss.org; messages dated Sat, 19 Nov 2005.

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

[44] CASSM: www.uclic.ucl.ac.uk/annb/CASSM/

[45] Clarke, S. (2004) Measuring API usability. *Dr. Dobb's Journal* Special Windows/.NET Supplement, May 2004 pp. S6-S9

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

[47] Clarke, S. (2005) Personal communication.

[48] http://blogs.msdn.com/stevencl/archive/2003/11/14/57065.aspx

[49] Carroll, J. M. (1990) *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Cambridge, Mass.: MIT Press.

[50] Blackwell, A.F., Burnett, M.M. and Peyton Jones, S. (2004). Champagne Prototyping: A research technique for early evaluation of complex end-user programming systems. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC04), pp. 47-54.

[51] Roast, C.R. (2002). Dimension driven re-design - applying systematic dimensional analysis. In J. Kuljis, L. Baldwin, and R. Scoble, editors, *Proceedings of the 14th Psychology of Programming Interest Group workshop (PPIG 14),* pages 173-185. Brunel University, 2002. http://www.ppig.org/papers/14th-roast.pdf

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

[53] http://en.wikipedia.org/wiki/Refactoring

[54] Thimbleby, H. (2004) User interface design with matrix algebra. *ACM Transactions on CHI*, 11 (2) 181-236

[55] Howes, A. & Payne, S.J. (1990). Display-based competence: user models for menu-based interfaces. *International Journal of Man-Machine Studies*, 33, 637-655.

[56] Blandford, A.E. & Green, T.R.G. (2001) Group and individual time management tools: what you get is not what you need. *Personal and Ubiquitous Computing*. Vol 5 No 4. pp. 213–230

[57] Grudin, J. (1983) Error patterns in novice and skilled transcription typing. In W. E. Cooper (Ed.) *Cognitive Aspects of Skilled Typewriting*. New York: Springer-Verlag.

[58] Salthouse, T.A. (1986): Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological Bulletin*, vol 9 no. 3 pp 303-319

[59] Ainsworth, S. E. (1999). The functions of multiple representations. *Computers & Education, 33*, 131-152.

[60] Blackwell, A.F. and Engelhardt, Y. (2002). A meta-taxonomy for diagram research. In M. Anderson & B. Meyer & P. Olivier (Eds.), *Diagrammatic Representation and Reasoning*, London: Springer-Verlag, pp. 47-64.

[61] Barnes, D. J. & Kölling, M. (2004) *Objects First with Java: A Practical Introduction using BlueJ*, 2nd ed. Prentice-Hall / Pearson Education