Cognition, Technology & Work (2002) 4:180–196

Cognition
Technology &
Work

Representations in Human–Computer Systems Development

D. Benyon

Napier University, Edinburgh, UK

Abstract: When system developers design a computer system (or other information artefact), they must inevitably make judgements as to how to abstract the domain and how to represent this abstraction in their designs. Over the years human–computer interaction, or more generally information systems design, has had a history of developing competing methods and models for both the process and products of its development. Various paradigms have been suggested, often trying to keep pace with the changing nature of the design problem; from batch processing to interactive systems to work situations and most recently to designing for household environments. It appears timely, then, to review the nature of the design problem that faces the developers of human–computer systems and to consider some of the impact that different representations and different conceptualisations may have on their activities. Green (1998) has suggested that a single model of developing human–computer systems is not desirable, instead arguing for a number of limited theories each of which provides a useful perspective. The aim of this paper is to place competing methods side by side in order to see their strengths and weaknesses more clearly. The central tenet of the paper is that different views of both the human–computer system design process and the different abstractions, or models, that are produced during the design process have varying degrees of utility for designers. It is unlikely that any single method or modelling approach will be optimal in all circumstances. Designers need to be aware of the range of views that exist and of the impact that taking a particular approach may have on the design solution.

Keywords: Design approach; HCI; Human computer systems; Models

1. INTRODUCTION

In order to produce any artefact, two major activities have to be undertaken; the designer must understand the requirements of the product and the designer has to develop the product. Understanding requirements involves looking at similar products, discussing with the people who will use the product their needs and analysing any existing systems to discover the problems with current designs. Development may include producing and evaluating a variety of representations until a suitable artefact is produced.

The term 'design' refers to both the process of developing a product, artefact or system and to the various representations (simulations or models) of the product which are produced during the design process. Designers need representations that will help them to understand users' requirements and to represent this understanding in different ways at different stages of the design. 'Design inherently consists of the formulation of models of possible states of affairs in the world' (Goel and Piroli 1992, p. 396).

Design is an ill-structured activity in the sense that goals and constraints are underspecified (Simon 1981). It often involves a variety of people and so teaching and learning from one another is important (Greenbaum and Kyng 1991). The relationship between scoping and understanding the problem and problem solving is complex and iterative (Schön 1983). Selecting suitable representations of the artefact that is being created is, therefore, important for exploring, testing, recording and communicating design ideas and decisions.

Although all design work shares some characteristics, there are important differences. One distinction between differing types of design is 'design as craft' versus 'design as engineering'. Engineering design is 'the use of scientific principles, technical information and imagination in the definition of a ... system to perform pre-specified functions with the maximum economy and efficiency' (Jones 1981, p. 8). Design as craft, on the other hand, relies much more on the tacit understanding which the designer has of both the problem at hand and the materials being used. A further distinction may be made between design as craft and design as creative process where the focus is much more on creating an enjoyable and stimulating user experience than it is on economy and efficiency.

Human–computer systems design (HCSD) may be taken as a general term for any system involving humans and information technologies or other 'information artefacts'. In HCSD the design as craft versus design as engineering

debate has been discussed by Dowell and Long (1989, 1998). The central tenet of the Dowell and Long argument (Dowell and Long 1998) is that they would like to see a move towards design as engineering in the development of human–computer systems (HCS). This is characterised, they argue, by four major elements; a commitment to shared models, values which guide the solution to a problem, symbolic generalisations (engineering principles) and exemplars that demonstrate effective use of models, values and generalisations. As Dowell and Long point out, unlike other areas of engineering, the development of HCS has few physically grounded concepts at its disposal. The bridge designer has well-developed models of the tensile strength of materials, the interaction of materials with environmental factors and the relationships between strength and weight. Such representations have been tried out many times in practice and have been shown to be useful and successful representations for bridge design. In human–computer system design there are a few physically grounded concepts dealing with motor limitations, auditory and perceptual constraints and so on, but most of our work is concerned with conceptual structures, with information presentation and exchange. In developing HCS we have to deal with what happens inside people's heads and here we can only make conjectures as to what the most suitable representations are. There is still considerable debate about what constitutes cognition (e.g. Lakoff and Johnson 1999; Gibson 1979; Hutchins 1995) and about the importance of cultural, historical and sociological aspects of design (e.g. Nardi 1996).

Dowell and Long (1998) appears as the 'target paper' in a special issue of the journal *Ergonomics*. Ten commentaries on this paper are presented by researchers in the field and a response to these provided by Dowell and Long (*Ergonomics* 1998). These different points of view indicate that HCS is far from being a discipline with clearly understood models, values, principles or exemplars. Indeed over the years competing methods and models for both the process and products of its development have been proposed. For example, in human–computer interaction (HCI), there is no universally agreed conceptualisation or theory; there is the question of whether we interact *with* the computer or we interact *through* the computer (Bench-Capon and McEnery 1989; Barlow et al 1989); there is an issue concerning the importance of the concept of a 'task' (Benyon 1992; Diaper and Addison 1992) and the task–artefact cycle (Carroll 1990). Other people (e.g. Fischer 1989; Storrs 1989) argue for a communication model of HCI in which there is 'an interplay between [human] mental processes and external computational and memory aids' (Fischer 1989, p. 45). Others emphasise the importance of context and the social dimension of HCI (e.g. Nardi 1996) and others a mixture of ethnography and distributed cognition. Some advocate an object-oriented approach to systems development (e.g. Booch 1994), scenario-based design (Carroll 1995) or ecological approaches (Vicente and Rasmussen 1992). Domain modelling has recently emerged as an important topic (Sutcliffe et al 1996; ASE 1998) and elsewhere concepts in software engineering (Parsons and Wand 1997) and 'ontologies' are a keen subject of debate (e.g. Guarino and Poli 1995; IJHCS 1997).

It appears timely, then, to review the nature of the design problem that faces the developers of HCS and to consider some of the impact that different representations and different conceptualisations may have on their activities. The focus of this paper is very much on models for design, and although it does touch upon a number of philosophies and empirical theories the central part of the paper is primarily pragmatic. Of course, having an effective theory to underpin a design approach is useful because it is the theory that enables the designer to observe phenomena. This paper looks more at how theories have been embodied in design methods than it does at the theories themselves.

Green (1998) opposes the idea of a single model of developing HCS by arguing for a number of limited theories each of which provides a useful perspective. The aim of this paper is to place competing methods side by side in order to better see their strengths and weaknesses. The central tenet of the paper is that different views of both the HCSD process and the different abstractions, or models, that are produced during the design process have varying degrees of utility for designers. In Section 2 a brief review of the concept of a model is presented. Section 3 provides a broad-brush classification of the different influences that have impacted HCS and Section 4 offers a discussion of these. The conclusion brings us back to the central debate about theories and paradigms resulting from the emergence of highly distributed computing environments and designing for 'non-work' settings.

2. MODELS

If we wish to think about HCI, and successfully engage in HCSD, we need to employ appropriate representations or models. Models are devices for understanding, communicating, testing or predicting some aspects of some system; they are the mediating artefacts of design. Models abstract some domain of interest by hiding some details so that the important aspects stand out. They are 'professional' languages that both constrain and focus a discourse by limiting the range of concepts that can be expressed in the language (Kangassalo 1983; Benyon 1997). Models provide a certain perspective on a domain by employing abstraction mechanisms that are reflected in the content and the structure of the concepts employed by the model.

Abstraction mechanisms involve combinations of categorisation, or classification (treating a class of objects as a

single object) and aggregation (grouping related things together and treating the group as a whole). Different forms of abstraction are appropriate for different aspects of design. For example, Kao and Archer (1997) identify three types of abstraction. Horizontal abstractions focus on a number of facets of the problem at a given level of detail, vertical abstractions present several levels of detail along a single dimension of the problem and general abstractions serve to link these other two together. They use this classification of abstraction to suggest various aids that could be provided for the designer.

Modelling is difficult. A model will be more or less effective for a given purpose according to the characteristics (both conceptual and physical) possessed by the model and the relationships between those characteristics, the systems or domain that is the focus of the model, the modeller and the recipient of the model. A model must possess the necessary structure and processing capability to fulfil its purpose. The analytic, explanatory and communicative power of a model arises from the structure, operations and constraints which that model is able to capture (Kangassalo 1983). A model must also have suitable physical characteristics such as an appropriate notation. Any representation needs to be considered in terms of its usability; for the purpose at hand, for the producers of representations and the consumers of those representations (O'Neill et al 1999).

It is important to recognise that models are the mediating artefacts of the design process. From the scale models of a car designer, to the CAD models of an architect, the mathematical models of a mechanical engineer or the diagrammatic models of a computer programmer to the storyboards of a film director or the notation of a song writer, we can see abstractions of the finished product being used as part of the design process; whether craft, creative or engineering. In order to understand where and when models are effective and how models fit in to the modelling process, we need to understand some principles of models.

2.1. Aggregation and Classification

The definition of a model given above is that it is an abstract representation of something that suppresses unnecessary detail. This suppression of unnecessary detail can be achieved in two ways. Firstly, a model can group related things together and represent just the aggregate object, thus suppressing details. Another technique is to represent a whole class of objects as a single object, thus suppressing details of the individual objects. The first of these techniques is known as aggregation and the second as classification. Together they produce abstractions.

Aggregation is the process of collecting together a number of characteristics of something and treating it as a single thing. For example, I might identify an aggregate item in a system as a 'user'. In fact any user will have all manner of characteristics, but for the purposes of designing an interface a designer might consider some generic set of attributes as constituting the aggregate 'user'. A more diligent designer might identify different types of users who have different characteristics – different purposes in using the system, different amounts of experience, etc. These different types share some characteristics with the generic 'user', but differ in others. Aggregates are common in information systems development; an entity is an aggregate of its attributes, an object in object-oriented modelling is an aggregate of its attributes and behaviours. Entity subtypes might be identified, or objects might be defined in an inheritance hierarchy to deal with the different flavours of the generic type.

Once characteristics have been collected together into aggregates, a classification has effectively been accomplished. Classification is the process of recognising that various objects share certain characteristics and can therefore be treated as a single thing. We are able to classify things in some way because we recognise that they have certain shared characteristics that are of interest for the purpose at hand.

Classification of objects means that we can represent and discuss a complex situation in simpler terms. As we deal with the more abstract concepts, we make generalisations about things and as we move towards the more concrete objects we make specialisations. Selecting appropriate classes or categories and identifying the defining properties which determine whether or not something belongs to a class is an area that has been studied by philosophers over many years. One of the most famous examples is Wittgenstein's (1953) discussion of the class of games – what are the defining properties which allow us to treat football, chess, card games, board games, ring-a-ring-a-roses and so on as a single class: games? The observer will find many overlapping properties, but not a single property which connects them all. More recently George Lakoff has discussed classification in his book *Women, Fire and Dangerous Things* (Lakoff 1987) and highlighted how different people classify things in different ways. Indeed the title of his book is chosen because these seemingly diverse things are classified as belonging to the same category by the Australian aboriginal language, Dyirbal.

2.2. Structure, Function and Purpose

In addition to using abstraction, models may focus on different aspects of the artefact being modelled. The modeller can look at the structure of the artefact, the functioning of the artefact or at the way the artefact changes when things happen to it.

A structural view of some artefact or system focuses on the main entities or objects that are in the system and how those objects are related. For example, the structural view

of a car would describe a car in terms of the main components – engine, gearbox, drive shaft, brakes and so on. A functional view focuses on how some substance, or some object, moves through the system. In the example of a car, we could look at how fuel flows though from the fuel tank to the engine, and how it is transformed into exhaust gases and flows out through the exhaust system. Another functional view might focus on the braking system and the flow of hydraulic fluids. The third view of a system focuses on the dynamics of the system, how it moves or changes from one state to another. This view is concerned with how the structure and functioning of the artefact are related and the behaviour of the system that results.

The three views of systems described above are complementary and interrelated. Notice how the different views interact. Certain functions can only occur when the system is in certain states and certain functions change the state of the system or of certain objects in the system.

The relationship between structure and function must be taken into account with the level of abstraction at which the object or domain is being viewed, since a structural view at one level of abstraction is a functional view at another. If the focus is on the car moving along a road, the functional view is of the movement of the car and its interaction with the road. Thus the structural view is of car and road as aggregate objects. If the focus is on the operation of the braking mechanism, the level of abstraction shifts to consider the structural relationships between objects such as brake disks, levers and so on and a functional view of how these work together to form a braking system. Vicente and Rasmussen (1992) provide a more thorough discussion of abstraction hierarchies. The appropriate view and the appropriate level of abstraction at which to describe some domain or system will depend on the purpose of the modeller.

At a given level of abstraction the designer can look at the system or domain from one or more of three perspectives. The intentional level of description is concerned with how the system relates to entities external to that system (at a particular level of abstraction). The conceptual or logical level focuses on the function and structure of the system – on its semantics. The physical level is concerned with how things work in a physical world. Evidence that these three levels are necessary come from a number of areas such as Rasmussen's consideration of mental models and HCI (Rasmussen 1986) and the philosophical arguments of Pylyshyn (1984) and Dennett (1989). Pylyshyn argues that what 'might be called *the basic assumption of cognitive science* ... [is] that there are at least three distinct, independent levels at which we can find explanatory principles ... biological, functional and intentional' (Pylyshyn 1984, p. 131, italics in original). He relates these terms to those used by other writers. For example, Newell (1982) calls them the device level, symbol level and knowledge level. Pylyshyn himself prefers the terms physical, symbol and representational (or semantic) level. In Dennett's terms these are the physical stance, the design stance and the intentional stance. Mappings between the levels are required in order to progress from one level of description to another.

Designers will move rapidly between different types and levels of abstraction during the design process, and different representations will be more appropriate at different times. At whatever level of abstraction the domain is modelled it is important to consider the appropriateness of our model in terms of the structure (and hence constraints) that it can represent, the functions that it supports and the structure/function relationship (the behaviour). In particular, it is important to consider the concepts that the model employs and the physical manifestation of the model.

2.3. Concepts in Models

Another important aspect of a representation is that the concepts that it employs constitute the 'material' from which that model is made. The focus, or object, of a representation is some domain or system that the modeller is interested in and which demonstrates (for the modeller) some stability and coherence. The concepts that the modeller chooses to use as the basis of the representation have a significant impact on the effective use that can be made of the model. This important distinction between the object of a model and the concepts from which it is constructed is often overlooked.

For example, if I perceive something in the world that I call a circle, then I can choose to model this in a variety of ways. An algebraic model of a circle, $x^2 + y^2 = z^2$, is not a model *of* algebra. It is a model *of* a circle *made from* algebra. This could be contrasted with a model of a circle *made from* programming constructs expressed in a language such as Logo (which may be represented as: To Circle, Forward 1, Right 1, Circle). A graphical representation of a circle is another representation. Notice the different structures, constraints and functions provided by these models of a circle and the different purposes to which they can be effectively put. The algebraic model allows me to manipulate the formula and to deal with different sizes of circle, but gives a very poor idea of its shape. The graphical representation immediately conveys the shape concept and the computer program lets me control things that would form circles. None of these is best in all circumstances; it depends on the purpose of constructing the model.

Consider the difference between an entity-relationship (E-R) model of a database and a relational model of a database. The E-R model is a graphical representation, allowing designers and users to see the overall structure of the database clearly. It can be used to identify areas of semantic ambiguity in the database by tracing through the

relationships in order to verify that these 'paths' through the data are consistent (Benyon 1997). The relational model does not facilitate this particular activity, but it does allow for structured browsing of the data using a query language such as SQL. The designer will need to develop both these types of model during the development of a database. During the development of these models the designer will use other representations such as functional dependency theory and normal form analysis to ensure that the models accurately reflect the user's understanding of the data in a domain.

2.4. Physical Form

In addition to considering the conceptual 'stuff' that a model is made of, we need to consider the physical representation – the notation. Green (1995) discusses the 'cognitive dimensions' of notations and how these can affect their usability. Some notations force the designer into making a 'premature commitment' to some object or concept before he or she is ready to do this. Some notations are difficult to change (they are 'viscous') and hence encourage designers to remain with a suboptimal design. Some notations are difficult to understand (e.g. diagrams with lots of crossing lines), while others can use colour to help clarify different semantics. For example, O'Neil et al (1999) discuss drawing task models on a whiteboard using different coloured pens.

As a final example, consider maps as models of a terrain. Maps are generic models which come in a variety of physical manifestations. Maps are abstractions from the domain (the terrain) which emphasise some features while suppressing others. If one wants to find a path from one village to another then a relief map (one that shows only the height of the land) is unlikely to be of much help since it does not include the concept of a path. Its purpose is not appropriate for the purpose at hand. If we have a map of the scale 1:50,000 this may be more suitable for the purpose of finding a path (but is less suitable for seeing the distribution of hills and valleys). The map may have been designed to show fields, fences, rivers and paths and so contains the required functions (the ability to follow a path on the map) and the necessary structure (the concepts of path, etc.) to fulfil its purpose. However, the map may be so poorly designed at the physical level (e.g. it uses black lines to show boundaries and paths and rivers) that it is not usable for its purpose.

2.5. Modelling Human–Computer Systems

The question for HCSD, then, is which representations should be used for which purpose. The domain of HCS is very rich; it is not a simple concept such as a circle. The HCS domain is a complex combination of structures and behaviours involving people, technologies, environments and purposes. We need to be confident that the representations which we use in HCSD have an appropriate structure and an appropriate physical representation so that they can serve their purpose. Moreover, since HCSD is concerned with developing HCS, the concepts which underlie any given representation must be concepts that are suitable for representing people and their interactions with the other people, artefacts and information which collectively make up the human–computer system.

The purpose of HCS design is to develop effective HCS. It follows that any model which we use must be oriented towards that purpose. Even with this declared purpose there are many levels at which we can view HCS. At the organisation level we might wish to consider the design of working practices and the impact of new technologies on organisations. At the environmental level, we might wish to look at health and safety issues or office layout. Another view at this level might be concerned with legal and ethical issues concerning HCS. We may want to look at information flow, at how knowledge is constructed and represented or how it is distributed through the system. At a physical level we could examine the use of colour or font style on the usability of systems, at the physical or cognitive demands that systems make on people. At a more detailed physical level we may consider hardware and software issues concerned with rendering accurate representations of objects or supporting various styles of interaction.

We also need to consider what we mean by 'effective'. The Dowell and Long engineering view sees effectiveness as the resource needed to learn and use a system. Vicente (1999) sees effectiveness as safety, productivity and health with respect to computer-based work. Like so many other features of HCSD 'effectiveness' is a term that has been interpreted and reinterpreted in different ways over the years.

Developing suitable representations, or models, of an artefact is fundamental to all design activities. The representations employed may be formal or informal, precise or vague and may be used for very different purposes within the overall design activity. Different representations will be more or less useful depending on the level of abstraction at which we which to view the domain of HCS. One of the skills of the designer is selecting an appropriate representation for the task at hand. Another is making good use of that representation. In the next section a review of the main methods and paradigms in HCSD is presented.

3. MODELS AND CONCEPTS FOR HUMAN–COMPUTER SYSTEMS DESIGN

Before looking at some generic methods and models in HCS design, it is worth spending a few paragraphs looking at a history of the discipline. The following is an extended

version of a similar history that appeared in Benyon and Imaz (1999).

Although the subject of HCI did not really enter onto the agenda until the early 1980s, the previous two decades had seen several attempts at developing methods, approaches and modelling techniques for information systems design. The early days of systems analysis and design evolved methods and models which focused how the existing system worked. Representations of the system, usually in terms of a flowchart, which provided a graphical representation of the existing system, were produced which led to a computer system and a file structure to support the particular application (Benyon and Skidmore 1987). Where systems were well understood and primarily cyclical in nature, such as monthly accounting or payroll systems, they could be computerised without too much trouble. However, these methods of analysis began to break down with the introduction of more interactive working, when other people wanted to use the data which had been gathered and stored for a specific application and when computers were being applied to less well-defined activities. During the 1970s systems analysis fundamentally changed following the publication of Ted Codd's seminal paper (Codd 1970; see also Codd 1982) on the relational data model. The 'data-centred' movement in information systems development looked to distance itself from current implementations (i.e. to be more abstract) and this led to dataflow diagrams (e.g. DeMarco 1979) and the E-R diagram (Chen 1976). The movement faltered with the development of many competing methodologies (Olle et al 1982, 1983) and finally gave way when the object-oriented (OO) paradigm, inherited from software development, began to be applied to analysis and design as well as to the construction of systems. Since the early 1990s, OO methods have been paramount (e.g. Booch 1994).

The concerns of HCSD, as we know it today, began in the sixties but were most clearly expressed in the late 1970s through the participative and socio-technical approaches to systems development, a movement which remains important today (Greenbaum and Kyng 1991; Karat 1991; Kyng and Mathiassen 1997). Methods and models encouraging user participation in design developed alongside models of users interacting with computers derived from task analysis (e.g. Moran 1981; Diaper 1989). As graphical user interfaces began to dominate HCSD, so task-based and OO methods followed and the rather difficult, grammar-based task models were replaced by graphical notations with the focus on user objects. These became embedded in more proscriptive HCSD methodologies (such as STUDIO; Browne 1994) and tool-supported systems (e.g. ADEPT; Johnson et al 1995). The somewhat different traditions of cognitive (systems) engineering (Hollnagel and Woods 1983; Rasmussen 1987) and computer-supported cooperative working (CSCW) have also contributed to the current position in HCS. All this took place as the philosophy of mind was developing with the everyday psychology and seven-stage model of interaction of Norman (Norman and Draper 1986). The strong cognitive psychology of the 1960s and 1970s had put the focus on user tasks and the detailed analysis of tasks deriving from the workflow analysis which had happened during the 1960s. Influential philosophical developments by, for example, Winograd and Flores (1986) and later Hutchins' (1995) work on distributed cognition, and the introduction of activity theory into HCSD (Bødker 1990; Bannon 1991), have changed the influence from cognitive tasks to situated action. More recently, context has become an important issue in HCSD through techniques such as scenario-based design (Carroll 1995) and contextual inquiry (Beyer and Holzblatt 1998) and further developments in cognition such as cognitive semantics (Lakoff 1987; Lakoff and Johnson 1999). The challenge for the future will lie in the design of information appliances (Norman 1999; Bergman 2000) and in the evaluation of usability in household and other non-work settings (Sloane and Van Rijn 2000).

Throughout the history of HCS development many models and methods for the design of such systems have been produced. In the next nine subsections we look at a number of generic models and methods which are used in HCSD and consider them from the perspective of the concepts that they employ (i.e., the 'stuff' that they are made of). This will help us to see the structures, functions and purposes that they support. It is not our intention to describe all the detailed instantiations of these generic models, but where appropriate we do comment on the usability of some of the physical manifestations of the modelling approach.

3.1. Data-Centred Models

Data-centred models represent the flow and structure of the data in a system. The two main models are the E-R model (Chen 1976) (representing structure) and the dataflow diagram (e.g. DeMarco 1979) (representing processing). A number of 'data-processing models' (Benyon 1997) describing the structure/function relationships such as state transition diagrams (STDs) and entity–life history (ELH) diagrams (Rosenquist 1982) are also used in most data-centred methods.

All of these models use as their basic concept the notion of a data item or data element. A data element consists of one or more symbols, a name (and sometimes a more comprehensive description of the meaning of the data item) and a context. The name, description and context ascribe the semantics to the data element. Data elements are generalisations of the actual and potential values which that data item can take – the domain of the data element.

Another level of abstraction in data-centred approaches

is provided by something that is usually known as an *entity*. An entity is an aggregation of data elements, expressing the semantics that certain data elements belong together. Once these groupings have been established, the designer can refer to the collection of data elements – the entity – by name, thus suppressing detail which would otherwise clutter the model. Relationships between entities – expressed in terms of the number of instances of one entity which are associated with the instances of another entity – can then be considered. The physical representation of entities and relationships is usually in diagrammatic form of a E-R diagram, but it can equally be represented as a relational model which facilitates different manipulations. Representing the domain in terms of data elements, how they are grouped together into entities and the relationships that pertain between entities describes the structure of the domain.

Another way to model the domain is to look at how data flows between processes. A process, or functional, model made of data concentrates on the data which is strictly necessary for processing to occur. Functional data models are abstractions of systems which strive to be as independent of the current system as possible; the idea is to look at the logical flow of data through a system which is required if the system is to achieve a certain purpose. This may be contrasted with a model of the physical system (such as a flowchart) which models the movement of physical objects such as documents and the control flow (i.e. the sequencing of actions) of particular implementations. Functional data models concentrate on the data processing that is strictly necessary for some process to function in a given domain. They show the data that is necessary for certain processes to happen.

Data-centred models have a theoretical basis which dates back to Langefors (1967) and Sundgren's (1975) work and have a basis in terms of semiotics (Stamper 1977). They are central to many information systems development approaches (Avison and Fitzgerald 1996). The concepts of data element, entity, relationship and dataflow focus attention on the information in a system. Focusing on data and information has been applied to HCI (Benyon, Green and Bental 1999), enabling designers to highlight the difference between designers and users models, where the interface fails to reveal an underlying conceptual structure and issues of distribution of information (Green and Benyon 1996). Data-centred models can be used as the basis for selecting interface metaphors (Braudes 1991) and for the development of interactive visualisations (Tweedie 1995).

3.2. Task-Based Models

Task models seek to represent the domain in terms of tasks and actions. The emphasis of task-based approaches is primarily on human tasks and the need to understand what people (rather than computers) have to do. Task models were developed because of the emphasis in HCI on people interacting with computers and the dominant information-processing paradigm of cognitive psychology.

Task-based approaches have as their basic concept the idea of a user 'task' which may be defined as 'a goal together with some procedure or ordered set of actions that will achieve that goal' (Amodeus 1994). Hence other basic concepts include 'goal' (a state of the environment or agent which is desired by the agent) 'operation', 'simple task', 'unit task' or 'action' (a task which involves no problem solving or control component) and 'plan', 'method' or 'procedure' (a sequence of tasks, sub-tasks and/or actions). Task-based approaches date back to the earliest exposition of HCI theory (Card et al 1980; Moran 1981) and continue to be popular today (e.g. Browne 1994; Lim and Long 1994).

The wide variety of task-based models, the alternative notations which are used and the different uses to which they are put have a significant impact on the usability of different manifestations. The notation used in task-based methods is either based on a grammar type of representation, particularly for more detailed task description languages such as TAG (Payne and Green 1989) and GOMS (Kieras and Polson 1985), or it is based on the structure chart notation. Structure charts are a graphical notation which represent a sequence of tasks, sub-tasks and actions as a hierarchy and include notational conventions to show whether an action can be repeated a number of times (iteration) and the execution of alternative actions (selection). Task-based models are able to show the sequencing, iteration and selection of sub-tasks and actions and to show the allocation of tasks to human or to computer. The grammar notations of methods such as GOMS often become unwieldy if applied to large systems, but do provide a certain rigour which may provide useful insight into the detailed design of an interaction (Gray et al 1990).

Task-based models are representations of the functioning of a domain. They concentrate on the mental and physical actions which users and systems perform given a particular design. Although they can be used in a more abstract way to describe generic tasks in a domain, task-based models do not show the flow of information through a system, nor do they represent the system's structure. Structure, where it is shown, is typically represented through object models.

3.3. Object-Oriented Models

In OO methods of systems development the domain is represented in terms of the objects which exist, the relationships between objects and the messages which are passed between objects. The proliferation in object-

modelling methods has resulted in some confusion over exactly what is being represented, how it should be represented and how to approach domain modelling using the OO approach. The best-known methods included Booch (1994), Rumbaugh et al (1991), Jacobson et al (1993), Shlaer and Mellor (1992), Coad and Yourdon (1992) and Wirfs-Brock et al (1990). Towards the end of the century the first three of these became united in the Unified Modeling Language (UML; http://www.rational.com/uml).

Many claims have been made for OO techniques, most notably that the approach leads to a more natural design. For example, Rosson and Alpert (1990) claim that 'A particularly attractive aspect of OO design is that it seems to support a better integration of problem and solution' (p. 361) ... while admitting that 'very little empirical evidence exists concerning the naturalness of objects as ways of representing problem entities' (p. 363).

Objects are the basic concept in OO approaches. Objects are defined as 'an encapsulation of attributes and exclusive services [behaviours]; an abstraction of the something in the problem space ...' (Coad and Yourdon, 1992). Davis (1993), like many OO theorists, stresses that objects correspond to real-world entities. The fact that objects encapsulate structure and behaviour is central to the distinction between objects and entities in data-centred methods. OO modelling is a model of structure – it is the objects that are paramount. The system's processing is defined in terms of the objects – different classes of object allow for different types of processing. The notation for object diagrams (i.e., the physical representation of the model) varied considerably from E-R diagrams to 'roundtangles' to represent objects to cloud type diagrams (Sully 1994) until the standardisation that occurred through the UML development.

The claims for the benefits of OO techniques include abstraction and encapsulation (otherwise known as 'information hiding'). Objects are viewed from the outside and users need not be concerned about how they are implemented. Objects can send and receive messages; they encapsulate the structure and processing which enables them to deal with those messages. Other features of the OO paradigm such as polymorphism (that different object classes will treat the same message in different ways), inheritance, late binding and tight cohesion relate more to the programming benefits than to any conceptual benefits of the approach.

OO methods are not based on any underlying theory of cognition or action. The philosophical basis for OO approaches is derived instead from the benefits that come from OO programming; non-functional decomposition (i.e., a structural decomposition as a reaction against the functional decomposition of such techniques as dataflow), reuse and inheritance. The fact that users interacting with graphical user interfaces use interface objects and the idea that users themselves think in terms of objects has led to the popularity of these approaches within HCSD and a number of HCSD methods have been produced (Redmond-Pyle and Moore 1995; OVID 1997).

The ideas of standard structures have been extended from simple objects to complex 'patterns' of both systems and interactions. Recent approaches to HCSD now often talk of pattern languages and research seeks to establish a definitive list of interaction patterns. Sutcliffe and Maiden (e.g. Maiden 1998; Sutcliffe and Maiden 1998) have developed object models of a number of generic systems which they tie in with generic scenarios. The NATURE system has a psychological underpinning in terms of categories and mental schemata.

3.4. Systems

Since the earliest days of information systems development, systems theory has been an important influence (e.g. Laszlo 1969). Systems theory has provided a number of useful concepts including ideas of control (feedback, feed-forward, etc.), which come from cybernetics (and which we return to in Section 3.8). Systems theory has also been manifest in a 'hard systems' approach and through socio-technical systems design, but it is with the 'soft systems' proposed by Checkland (1981) which is more the concern of HCSD. The soft systems approach argues that instead of teasing HCSD apart into humans, tasks, interfaces, computers and so on, it is better to begin with a more basic concept: a system. A system is a more or less complex object which is recognised, from a particular perspective, to have a relatively stable, coherent structure (Checkland 1981). For example, I might wish to consider a human interacting with a computer as a system. Another person might wish to consider the computer as a system in its own right. Another person might wish to consider the users of networked computers distributed across the globe as system. Checkland (1981) emphasises the importance of human-activity systems as a particular class of systems that, because of the human component, cannot be simply engineered.

The concept of a system can be applied at many different levels of abstraction and from many perspectives. Checkland (1981) stresses the need to declare explicitly, as part of the system definition, the perspective (or *Weltanschauung*) from which the phenomenon is being considered as a system and the aspects of the system which are considered to be stable and coherent. This also results in identifying the system's boundary.

Systems theory recognises that all systems are composed of other systems and exist within wider systems. Systems interact with other systems. Systems interact with their subsystems, with their super-systems and with systems at the same level of abstraction. The interaction of a system's

component subsystems results in the system having properties which *emerge* from the relationships existing between its subsystems. In other words, systems possess properties which are not possessed by any of its subsystems.

Thus the main concepts from systems theory are the system concept itself, the declaration of the perspective from which the boundary of the system is determined and the emergent properties possessed by that system. The notion of a system underlies many more specific approaches and has been applied extensively in more general information systems design (e.g. Avison and Fitzgerald 1996). Physically, system models are often realised as 'rich pictures' which show the main features of the system – the customers, actors, transformations, Weltanschauung, owners and environment – in diagrammatic and pictorial form.

3.5. Semiotics

Semiotics is the study of signs and has been exploited in many areas of human endeavour such as linguistics, anthropology, psychoanalytic psychology and literary theory (Sturrock 1986). This wide range of applications has resulted in a number of different views on the role and structure of semiotics. In computer science and HCSD semiotics has been left relatively untouched. This may seem surprising since semiotics deals with the transmission and interpretation of signs – something fundamental to HCI. The exceptions to this include a detailed semiotic view of HCSD (Andersen 1990) which aims to develop useful taxonomies of computer-based signs and an approach known as semiotic engineering (de Souza 1993).

One of the weaknesses of semiotics (which is surely also one of its strengths) is the very general concept of a sign. Indeed semiotics has been called the mathematics of the social sciences as this concept is so broadly applicable and has been applied in many different ways at many different levels of abstraction. However, it is important to remember that all signs must be culturally determined and, very often, individually determined. That is to say, individuals and people from different cultures (whether ethnic, educational, work and so on) will interpret signs to stand for different things.

Signs work by bringing together some 'system of signification' with some signals. We can think of the signals as the syntactic part and the system of signification as the semantic part of the sign. Once these two are brought together the sign can represent, or stand for, something else; it can mean something. Such a simple characterisation of the semiotic process masks a number of issues such as the 'infinite semiosis' or 'hermeneutic circle' that says that things can only be interpreted in terms of things that have already been interpreted which in their turn were interpreted and so on.

The issue about where this 'system of signification' comes from is also important. Ultimately it has to be supplied by the person, or agent, that receives the signals. But some of it may be carried by the signals, particularly if they are themselves complex signals such as a building in a city. It is this relationship – between the signal(s) and the things signified – which is a sign. Eco (1976) emphasises that a sign is always a sign-function: the *sign-vehicle* for the syntactic part of the sign (the signals once they are associated with a semantic system) and *sign content* is the semantic part of the sign. 'Symbol' is used by a number of authors (e.g. Rasmussen 1986) to indicate the thing which the receiving system 'understands' – i.e., the sign contents.

Thus the basic concepts underlying semiotics are: the signifier (or sign-vehicle), which comprises the signals when placed in a relationship with a system of signification; the signified (or sign content), which is the (conceptual) entity associated with a signifier when put in relationship with a system of signification; and the sign (or sign-function), which is the relationship between the signifier and the signified. Both the sign-vehicle and the sign content have two components: a *substance* representing what each is and various different *forms*, or representations which it may take.

Although HCSD has not seen many applications of semiotics, the time for an application of these concepts may now have arrived as we try to develop 'intelligent' interface agents, as we increasingly consider the different media of presentation and in order to deal with people interacting with multiple embedded systems. The ability of an agent to receive and interpret signs leads to a number of questions which we can apply to agents. For example, we can ask: What signals are detectable? What semantic systems does the system have access to? What ability does it have to represent the context of the interaction? What goals, motivations and purpose are represented in the system and to what extent can these be used in the system's reasoning? And so on (Benyon 1993).

Semiotics has not yet been made available to HCSD developers in the form of a method or modelling technique, which may explain its lack of uptake in the discipline. Our purpose here is not to advocate which method, or concept is best. Semiotics represents a different conceptual foundation for thinking about HCSD.

3.6. Activities in Context

During the mid–late 1980s and into the 1990s, HCSD has witnessed a rise in methods and approaches to design which emphasise the need to see human activities in context. Suchman's (1987) criticism of the concept of a 'plan' – a pre-mediated sequence of actions directed towards a well-defined goal – and her argument that human action was situated within a context led to a number of approaches to

HCSD which sought to focus on human activities. Two important concepts are introduced in such approaches. The concept of an 'activity' consists of a subject (one or more individuals), an object (held by the subject and motivating the activity), actions (goal-directed and conscious processes that must be undertaken to fulfil the object) and operations (former actions that have become routine and unconscious with practice) (Nardi 1996). An activity is ... 'a system that has structure, its own internal transitions and transformations, its own development' (Leont'ev 1978, p. 50). An activity is directed toward a certain object (i.e., a purpose or goal) and is mediated by one or more artefacts (which may include pieces of software, 'thinking' artefacts such as language and so on). Activities can only be understood given some knowledge of the object, the motive, behind the activity and importantly need to be seen within a cultural and historical context; the term CHAT (cultural historical activity theory) is often used to emphasise this. The essential mediating nature of artefacts in any interaction has led Kaptelinin (1996) amongst others to talk about computer-mediated activity instead of HCI.

The second concept is that of 'context', although this is much less formally defined. Nardi (1997) describes the importance of context as follows: 'it's not just the users themselves, but the total environment they are working in, including other technical and cultural resources that are important'. The emphasis is on understanding user actions within this context. For others, the concept of 'context' is a combination of notations which aid envisionment, design rationale, scenarios and more formal notations needed to communicate contextual information to designers (Cockton et al 1996). Still the issue of context as something that surrounds an activity against context as something that holds the activity together is open for debate.

Activity theory has its origins in the work of the Russian psychologist Vygotsky, beginning in the 1920s, but has only recently been recognised in the Western scientific community (e.g. Leont'ev 1978; Bødker 1990; Bannon 1991). These approaches take into account the context of the work and concentrate on the placing of users within a work system. Because of this contextual nature of activity theory, methods of systems analysis are often based on ethnographic studies of users in their work situation. Activity theory forces designers to consider the structure and level of human interaction in an activity and the level at which support is provided. Designers need to attend to the social interaction surrounding computer use and the interaction between individual and group objectives. The meaning of interactions is a function of the activity in which these interactions occur. It is not possible to completely understand what these interactions mean without the context.

Contextual inquiry (Beyer and Holzblatt 1998) is a related method in that it emphasises understanding context. Others which emphasise understanding context include participative and cooperative design approaches (see Greenbaum and Kyng 1991).

Approaches which emphasise understanding and designing activities in context focus on the behaviour of a system – the relationships between structure and function – rather than on independent views of each of these. There is little formality in the physical representations used within activity theory itself, but some of the applied methods such as contextual inquiry do provide notations to help in the analysis of activities in context. Kaptelinin et al (1999) have made the ideas of activity theory more available through a checklist and Engström and Escalante (1996) use a triangular representation of person, artefact and object in order to highlight the composition of activities.

3.7. Scenarios and Use Cases

Although scenarios and use cases came from different roots, they have enough similarities for them to be considered together as a generic model of HCS. Use cases came out of OO analysis and design, particularly Jacobson (Jacobson et al 1993), whereas scenarios have been used in HCSD in various forms for some time. Rosson and Carroll (1995) bring these two traditions closer together in that their scenarios and OO design progress in tandem. In Preece et al (1994) scenarios are considered alongside sketching techniques, storyboards and snapshots as 'envisioning' methods for systems development. Scenarios and use cases seek to describe 'an envisioned task from a user's perspective' (Rosson and Carroll 1995, p. 249). The principle underlying using scenarios and use cases in design is that they make user interactions with systems concrete. They are closely allied with approaches that emphasise contextual design (discussed in Section 3.6), but additionally provide the basis for making the rationale behind designs explicit. Claims analysis (e.g. Carroll and Rosson 1992) and design rationale (e.g. Maclean et al 1991) can be explicitly stated when there is a concrete scenario to describe. Exactly why certain design decisions have been taken, the options that were considered and the criteria which were used to decide between options can be recorded and documented.

The concepts underlying scenarios and use cases are user actions when interacting with a particular device. In this sense, scenarios deliberately use concrete, rather then abstract, representations. The philosophy on which this is based is summed up by Rosson and Carroll (1995) as follows: 'this belief is founded on the general observation that humans excel at reasoning about concrete situations' (p. 270). Jacobson (1995) describes how use cases should present a 'black box' view of the system. A use case model defines the system's behaviour and is developed alongside, and orthogonal to, an object model. Jacobson employs a graphical representation showing the interaction between

entities outside the system ('actors') and the use cases which are inside the system. When an actor uses the system, the system performs a use case and hence the use cases describe the complete functionality of the system. Jacobson's use cases are a more formal representation than scenarios and can be used throughout the development process, so that a high-level case can be traced through the system to the code that implements it.

Scenarios can be used throughout the design cycle – from gathering 'user stories' (Imaz and Benyon 1999) to conceptual (abstract) scenarios, concrete scenarios and finally to use cases that can be used for implementation. From user stories as requirements to scenarios for design and use cases for implementation emphasises how a single model in various states of formality can be used throughout the development cycle. There can be little doubt about the importance of scenarios and related ways of envisioning designs, but there are few firm foundations that designers can use, apart from broad guidelines, in order to structure their use of scenarios. As with the models described in Section 3.6, scenarios focus on behaviour rather than on structure or functions. Physically, scenarios may be manifest as short pieces of text, working prototypes or as sketches and storyboards.

3.8. Work Systems and Cognitive Engineering

Cognitive engineering is the term for HCS development preferred by a number of researchers (Hollnagel and Woods 1983; Rasmussen 1986; Dowell and Long 1989). Although cognitive engineering came from the domain of process control, and earlier expositions (e.g. Rasmussen 1987) may have tried to distance cognitive engineering from HCI, this is no longer the case. The principle underlying cognitive engineering is that when designing computer systems or any other 'cognitive artefact' we are developing a complete cognitive system. 'A joint cognitive system means that the system boundary encloses a natural cognitive system – a human being – as well as an artificial cognitive system' (Hollnagel 1997, p. 40). Seeing the whole as a joint cognitive system enables designers to recognise that this system is more than the sum of its parts; it has emergent properties (cf. Section 3.4).

Although the work developed separately, the ideas of distributed cognition (Hutchins 1995), which recognises that by designing a cognitive artefact it is not possible to simply replace human intelligence in a system, seem to underpin a cognitive engineering approach as does the domain-oriented approach of Fischer (1989, 1998). Designers need to recognise that by developing cognitive artefacts the nature of the cognitive processes in a system are changed. Hollnagel's approach utilises cybernetic theories of control (such as feed-forward and feedback) in addition to concepts such as 'amplification'.

Dowell and Long (1989) and Long and Dowell (1998) also use the term 'cognitive engineering' in their discussion of HCI. Long and Dowell (1998) present their view of cognitive engineering in terms of the 'dualism' of work system and domain. On the one hand is a combination of people (or in more general terms, agents) and devices which constitutes a (joint) cognitive work system. On the other is a domain: 'an abstraction of the real world'. The aim of cognitive engineering is to maximise the performance of this system with respect to this domain. Their conception highlights the need to see people using devices as a system. It recognises that it is only within this whole cognitive system that agents can formulate intentions to change the state of the domain. By opposing the work system and the domain, we gain the opportunity to engineer the system. Long and Dowell use this conception to define and measure the performance of the work system with respect to the domain. They are able to discuss task quality and provide definitions of the otherwise somewhat nebulous HCI concepts of learnability and usability in terms of the costs incurred by the different agents and devices in the work system.

The basic concepts underlying cognitive engineering are those of a joint cognitive system and a domain. In Long and Dowell's terms, the designer needs to consider design as the development of agents and devices and abstractions; these joint cognitive systems are the basic unit of analysis for cognitive engineering. For Hollnagel, it is designing for complexity, and designing to manage that complexity, that is important rather than trying to design complexity out. Lim and Long's (1994) methodology for HCI design is a concrete manifestation of the approach which concentrates on a functional (task-based) view of the joint cognitive system. Other approaches offer concepts and principles – a structural view of cognitive systems – but no concrete methodology.

An important approach to HCSD that has emerged from the area of cognitive engineering is 'the Riso genotype' (Vicente 1998, 1999), which has led to ecological interface design. Flach (1995) provides a number of perspectives on the issues and includes chapters by others originating from the Riso National laboratory, including Vicente and Rasmussen and Pejtersen. There is much discussion over the similarities between the psychology of Gibson (1979) and designing systems that afford certain activities. From our perspective, however, the most important feature is probably the concept of a means–end way of describing the domain. The 'stuff' of the representation is the relationships between means and ends. At the lowest level this is expressed in terms of some physical causes and effects and at the highest it is concerned with purposes and the effects that they have on the domain. The means–end hierarchy has five levels which may be mapped onto the physical, design and intentional levels discussed in Section 2.4.

Rasmussen and Pejtersen (1995) use this analysis to map out different work situations.

3.9. Constructional and Interactional Models

Most of the models presented in the previous sections (with the exception of OO and data-centred models) have been concerned with the analysis and design of systems as opposed to their construction. The designer cannot give a scenario, or an activity description to a programmer and ask her to implement it. In a recognition of this, much work has continued on finding appropriate models which software developers can use. The occasional coming together of researchers in user interface systems engineering and in HCSD (e.g. Benyon and Palanque 1996; Benyon et al 1997) has demonstrated the huge gulf that exists between the demands of good analysis and the demands of good construction.

One area of debate concerns the appropriate software architecture for HCS. The concerns of software architectures are centred round reusability, maintainability, integrity and so on. Although these concerns may seem a long way away from understanding user intentions, or the meaning behind their activities, unless the high level models of analysis and design can be translated into the models of software construction, then we will fail to develop effective HCS. The European Amodeus project (Amodeus 1994) did bring together these differing perspectives, looking at models of interaction from both software and cognitive positions. Models based on the ICS framework (Barnard 1991; Barnard and May 1999) focus on a cognitive interaction model was used alongside architectures such as PAC, resulting in an 'agent-based' architecture, PAC-Amodeus (Coutaz et al 1996). Software architectures (reviewed in Coutaz, et al 1996) distinguish between the conceptual (or abstract) and presentation views of systems at 'every level of abstraction' (Coutaz et al 1996, p. 194). Inevitably detailed in their concerns, architectures do reflect the purpose, conceptual and physical levels of models described in Section 2.

Other models which seek to bridge the gap between analysis/design and construction include the Interactive Cooperative Objects (ICO) method, based on objects and Petri nets (e.g. Palanque and Bastide 1996). This approach seeks to use the same formalism to represent user task structures as it does to represent internal software structures. The method is manifest as a number of related task and system diagrams. UAN (Hix and Hartson 1993) represents the main objects of interest (mouse buttons, pointers, etc.) and the tasks which are required to interact with a system (click mouse, move pointer, etc.). UAN uses a structured grammar notation and is most applicable at a detailed level of design, but can be effective in representing more general task structures.

The concerns of the models discussed in this section are based around finding a formalism that allows the construction of software systems. With the popularity of OO implementation, most of them use the concept of an object. In order to accommodate the concerns of HCSD, many use some form of task representation, usually more formally expressed as, for example, a Petri net. The distinction between abstract and presentation views of the object or system is also emphasised. These interaction models stress the behaviour of the system. Many such models use a graphical representation with the added rigour of employing methods with formally defined semantics.

3.10. Discussion

In this quick tour through the representations used in HCSD, we have tried to arrange and classify the models, methods and approaches in a relevant way – relevant, that is, for our purpose. The aim has been to include most of the main influences on HCSD, but many, many different techniques and models have been omitted. The classification above is just one example of how these methods could be organised. It is perhaps surprising that a real method has not emerged from the computer-supported cooperative work (CSCW) area. Issues of awareness and distribution of control are particularly important here and some headway is being made by, for example, Viller and Sommerville (1999). Their method seeks to capture the results of ethnographic studies in a form suitable for systems designers. Distributed cognition as a fundamental approach has also been omitted because real methods have not appeared based upon it. The ideas are clearly crucial to our understanding of HCSD and Green and Benyon (1996) do show how representing the distribution of information in a system can be useful.

We have seen that the various approaches described in the previous section have arisen from a variety of experiences and theories. Some are based on practical experience. For example, the object concept is not derived from any theory of human cognition or action: it was originally a programming construct. The concept of a 'task', on the other hand, is derived from information-processing psychology and the concept of an activity has developed from activity theory. Some of the concepts are well formed (e.g. the entity construct in the data-centred methods or the concept of a sign in semiotics) whereas others are much more vague (e.g. context) and others include a range of alternative conceptualisations (e.g. the scenario concept). Some of the concepts used by the different approaches are very similar (e.g. the system concept appears in systems theory, activity theory and in cognitive engineering) and others are closely related (e.g. the idea of task and the idea of scenario). There are, in addition, similarities between the underlying theories. For example, when the emphasis is

put on a cognitive system consisting of individuals and artefacts then we would be talking about distributed cognition, but when the emphasis is mainly put on a group of people, working and communicating through artefacts, we would be talking about activity theory.

Systems theory, activity theory and distributed cognition take a holistic, systems view of the world. An activity is 'a system that has structure'. Similarly for distributed cognition it is the overall system (which includes many people and many information artefacts) which has a goal; the goal of this activity of flying a plane is the 'successful completion of a flight' (Hutchins 1991). In terms of activity theory this would be expressed as: the *object* (more or less equivalent to the system's purpose or goal) of the activity is the successful completion of a flight. In systems theory we perceive and define systems which have an ongoing purpose and a coherent structure.

An important distinction may be made between objectivist views of 'the world' and subjectivist views. Whereas approaches emphasising activities in context are openly subjectivist, the object paradigm takes a highly objectivist view. Designers are encouraged to 'discover' objects, objects are related to objects in the real world, and so on. In data-centred methods, on the other hand, subjectivity has been emphasised more; an E-R model is a subjective representation of a perceived situation (Lewis 1994). Semiotics is fundamentally concerned with interpretation of signs and with the subjective nature of the 'hermeneutic circle' – signs can only be interpreted in terms of the other signs, which in their turn were interpreted in terms of other signs, and so on. Some of the approaches try to avoid this distinction by developing rich, subjectivist representations, and then embody these in concrete, objectivist scenario or task descriptions. We have also seen the distinction between concepts based on well-formed theories (e.g. activity, entity, system, sign) and others which are based on 'spontaneous' philosophies (e.g. scenario, object, context). Table 1 provides a summary of the main features of the different approaches that we have considered.

There is inevitably a dilemma here for HCSD as our purpose is to construct HCS. Hence we have to embody the results of our analysis in concrete artefacts at some point. For anthropology, for example, this is not a problem as the purpose is to understand and describe, not to redesign. We have seen the soft approaches to systems development emphasising envisionment, participation and cooperation. Others (e.g. Dowell and Long's cognitive engineering) take a harder approach in which concepts of usability and learnability can be objectively defined and measured.

The complexity of HCSD requires us to employ a range of models in order to gain the variety of insights that are necessary if we are to design successful HCS. At different points in the design process, different models will be more or less useful for designers. The models and methods reviewed in Section 3 may all have a place to play. For example, OO methods are desirable in order to construct systems using OO systems, but may have problems at the analysis/design stage since they are not as rigorous as entities in E-R modelling. Scenarios, tasks and use cases seem appropriate for evaluating alternative designs and for envisioning future interactions, but may be too welded to current implementations to provide the level of abstraction necessary for creative design to occur. Data-centred models focus on the exchange of signals between the agents and devices which constitute that system. Focusing on data or signs is more abstract, more device independent and makes the knowledge assumed by different agents and devices explicit. We need to consider the social construction of interaction and the use of language and thought in situations and the physical characteristics.

4. CONCLUSION

Finding the most appropriate representations for developing HCS is essential for effective design, but remains difficult while different conceptions of HCSD exist. In the review presented here, we can see different traditions emphasising aspects that are seen to be most important in

Table 1. Main features of approaches to HCSD

	Concepts	Structure/function/purpose	Level of abstraction	Physical form
Data-centred	Entity, relationship, dataflow	Structure and function	Medium to low	ER diagram, DFD
Task-based	Task, plan, operation	Purpose (goals) and function	Medium to low	Structure diagram, grammar
Object-oriented	Object	structure	Medium, low	Unified Modeling Language (UML)
Systems	System, CATWOE elements	Purpose, structure	High	Rich picture
Semiotics	Sign	Structure	Low	
Activities	Activity, context	Purpose (object), structure	Medium–high	Activity checklist, triangular representation
Scenarios	Scenario, use case	Function	High to low	Storyboards, etc., use case diagrams
Cognitive engineering	Cognitive work system, means–end, domain	Purpose, structure	High to medium	Various diagrams (Rasmussen)
Constructional	Input/processing, output	Structure and function	Low	Many varieties

that particular area. Hence the tradition of task analysis where people are mostly concerned with a person using a computer. The ideas of activity and distributed cognition have arisen from studying collaborative work and cognitive engineering that has arisen from the development of complex control systems. We will soon have to face up to issues of designing for mobile devices, for intelligent systems and for networks of embedded systems.

With the ubiquity of 'information appliances' (Norman 1999) or information artefacts (Benyon et al 1999), the single-person single-computer view of HCSD becomes inadequate, a point recently emphasised by Hollan et al (2002). The ubiquity of computing means that we need to design for people surrounded by information artefacts. People are no longer simply interacting with a computer – they are interacting with people using various combinations of computers, information artefacts and media. As computing devices become increasingly pervasive, adaptive, embedded in other systems and able to communicate autonomously, the human moves from outside to inside an information space. In the near future the standard graphical user interface will disappear for many applications, the desktop will disappear, the keyboard and mouse will disappear. Information artefacts will be both embedded in the physical environment and carried or worn by people as they move through that environment.

This conceptualisation of HCSD is illustrated in Fig. 1. The basic ontology consists of activities, people (or artificial agents), information artefacts the domain. The aim is to move away from the dualism of cognitive system and domain and the focus on gulfs of execution and evaluation between a person and a computer. The figure tries to illustrate that these four things are intrinsically intertwined and it is this interlinkage that supplies or creates the context for action. People undertake activities in an activity space, but they are constantly moving between the activity space and the information space – the conceptual side of the information artefact (consisting of data,) being the information space and the perceptual side (providing a view onto that conceptual structure) existing in the activity space. There is also unlikely to be a simple one-to-one relationship between any of the components.

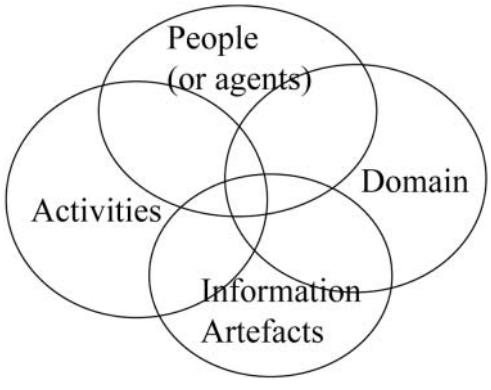

Fig. 1. Basic ontology of HCSD.

The notion that we can see people as existing in, and navigating through, an information space (or multiple information spaces) has been suggested as an alternative conceptualisation of HCSD (Benyon and Höök 1997). Looking at HCSD in this way means looking at HCSD as the creation of information spaces (Benyon 1998b). *Navigation of information space* is not (just) a metaphor for HCI. It is a 'paradigm shift' that changes the way that we look at HCSD. The conception has influenced and been influenced by new approaches to systems design (Munro et al 1999), usability (McCall and Benyon 2002) and information gathering (Macaulay et al 2000).

This change of view has many resonances with recent work by Hollan and Hutchins on distributed cognition (Hollan et al 2002) and with the ecological approach to design. 'Ecological interface design focuses on matching the structure in the interface to the natural constraints of the work domain in a way to inform and guide the operator as he navigates within that work domain' (Flach 1995, p. 10). Importantly, the move is away from the idea that we can maximise performance of some work system with respect to some domain. It is on recognising that some area of activity (something which some observer perceives as a system) consists of people (or artificial agents), information artefacts, activities and domain objects. It is this combination that is the unit of analysis and design, and it is the combination we seek to design so that the combination, as a whole, is effective in the pursuit of its purpose.

Navigation of information space is a new paradigm for thinking about HCSD, just as direct manipulation was a new paradigm in the 1980s. Shifting the paradigm changes the way you think about things. Computers still compute even though we have an OO paradigm instead of one based on functional decomposition. It happens that people believe that thinking about computing as an OO system helps to develop better systems. The metaphor that underlies a conception of HCI is crucial to the effectiveness of a that view for particular purposes (Benyon and Imaz 1999). Navigation of information space suggests the view that 'people are navigators' and encourages us to look to approaches from physical geography, urban studies, gardening and architecture in order to inspire designs. The changing models in HCSD can be seen as changing paradigms for understanding our discipline – or at least the design issues that we face. We might expect the direct manipulation, WIMP interface and task-based analysis and design to be superseded by the 'indirect management', delegated approach of agent-based interaction (Kay 1990). The passive retrieval of information from systems will give way to a more active involvement of people within their 'information space'. Attention will shift from application-based systems to domain-oriented environments (Fischer

1989, 1998). In such a situation, our metaphors for systems development must similarly shift. Benyon (1998) presents a conception of cognitive engineering as the creation of information spaces. Such a view encourages us to look to the designers of physical, geographical spaces – architects, city planners and the like – to help us understand our discipline and looks to spatial cognition and theories of navigation for a theoretical grounding (Spence 2000). In the design of physical space we have seen a move away from a utilitarian view of engineering towards a recognition of the social, cultural and political environment which people inhabit. Postmodernism has taught us that engineers cannot dictate the nature of space. It is people who produce spaces (Lefebvre 1983). Elsewhere (Benyon and Höök 1997) we provide a variety of alternative metaphors for thinking about information spaces – as a city, as an ocean, as a wilderness and so on, each of which enables us to concentrate of different aspects of user activities.

The analogy with HCSD is that design is concerned with creating information spaces which consist of agents and devices that represent things which are meaningful to people. People engage in cognitive, personal and social activities within these information spaces. Not only will people influence the design of these spaces, but we can expect a degree of autonomy as artefacts adapt to evolving situations. The concepts and representations that we need for designing such experiences may take us a long way from where we are now.

References

Amodeus (1994). Online at http://www-lgi.imag.fr/Les.Groups/IHM/AMODEUS.

Andersen PB (1990) A theory of computer semiotics. Cambridge University Press, Cambridge, UK.

ASE (1998). Special issue: domain modeling for interactive system design. Automated Software Engineering 385–464.

Avison D, Fitzgerald G (1996) Information systems development. McGraw-Hill, New York.

Avison D, Wood-Harper T (1990) Multiview methodology. Alfred Waller (McGraw-Hill), New York.

Bannon L (1991) From human factors to human actors. In Greenbaum J, Kyng M. (eds). Design at work: cooperative design of computer systems. Erlbaum, Hillsdale, NJ.

Barlow J, Rada R, Diaper D (1989). Interacting WITH computers. Interacting with Computers 1(1):39–42.

Barnard P, May J (1999). Representing cognitive activity in complex tasks. Human–Computer Interaction 14(1,2):93–158.

Bench-Capon TJM, McEnery AM (1989). People interact through computers not with them. Interacting with Computers 1(1):31—38.

Benyon DR (1992). Task analysis and systems design: the discipline of data. Interacting with Computers 4(2):246–259.

Benyon DR (1993). A functional model of interacting systems: a semiotic approach. In Connolly JH, Edmonds EA (eds). CSCW and AI. Erlbaum, London, pp 105–126.

Benyon DR (1995). A data-centred approach to user centred design. In Nordby K, Helmersen PH, Gilmore DJ, Arnesen SA (eds). Human–computer interaction: INTERACT-95. Chapman & Hall, London, pp 197–202.

Benyon DR (1996). Domain models for user interface design. In Benyon DR, Palanque P (eds). Critical issues in user interface systems engineering. Springer, Berlin, pp 3–19.

Benyon DR (1997). Information and data modeling. McGraw-Hill, Wokingham, UK.

Benyon DR (1998). Cognitive ergonomics as navigation in information space. *Ergonomics* 41(2):153–156.

Benyon DR, Höök K (1997). Navigation in information space: supporting the individual. In Proceedings of INTERACT '97. Chapman & Hall, London, pp 39–46.

Benyon DR, Palanque P (eds) (1996). critical issues in user interface systems engineering (CRUISE). Springer, Berlin

Benyon DR, Skidmore SR (1987). A tool-kit approach to information systems development. Computer Journal 31(1):2–7.

Benyon DR, Kilgour A, Sandblad B (1997). Integrating HCI and software engineering. In proceedings of INTERACT '97. Chapman & Hall, London, pp 691–692.

Benyon DR, Green TRG, Bental D (1999). Conceptual modelling for human computer interaction, using ERMIA. Springer, London.

Bergman E (ed) (2000). Information appliances. Morgan Kaufmann, San Francisco.

Beyer H, Holzblatt K (1998). Contextual design: designing customer-centred systems. Morgan Kaufmann, San Francisco.

Bødker S (1990). Through the interface. Erlbaum, Hillsdale, NJ.

Booch G (1994). Object-oriented design with applications. Benjamin/Cummings, Redwood City, CA.

Braudes RE (1991). Conceptual modeling: a look at system-level user interface issues. In Karat J (ed). Taking software design seriously. Academic Press, New York, pp 195–208.

Browne D (1994). Structured user design interaction: STUDIO methodology. Prentice-Hall, Englewood Cliffs, NJ.

Card S, Moran TP, Newell A (1980). The psychology of human–computer interaction. Erlbaum, Hillsdale, NJ.

Carroll JM (1990). Infinite detail and emulation in an ontologically minimised HCI. In Chew JC, Whiteside J (eds). Empowering People: CHI'90 Proceedings. ACM Press, New York.

Carroll JM (1995). The scenario perspective on system development. In Carroll JM (ed). Scenario-based design: envisioning work and technology in system development. Wiley, New York, pp 1–17.

Carroll JM, Mack RL (1985). Metaphor, computing systems and interactive learning. International Journal of Man–Machine Studies 22:39–57.

Carroll JM, Rosson MB (1992). Getting round the task–artefact cycle: how to make claims and design by scenario. ACM Transactions on Information Systems 10:181–212.

Checkland PB (1981). Systems theory, systems practice. Wiley, New York.

Chen PP-s (1976). Towards a unified theory of data. ACM Transactions on Database Systems 1(1):9–36.

Coad P, Yourdon E (1992). Object-oriented analysis (2nd edn). Prentice-Hall, Englewood Cliffs, NJ.

Cockton G, Clarke S, Gray P, Johnson C (1996). Literate development: weaving human context into design specifications. In Benyon DR, Palanque P (eds). Critical issues in user interface systems engineering. Springer, Berlin, pp 227–248.

Codd EF (1970). The relational model of data for large shared databanks. Communications of the ACM 13(6):377–387.

Codd EF (1982). Relational database: a practical foundation for productivity. Communications of the ACM 25(2):109–117.

Coutaz J, Nigay L, Salber D (1996). Agent-based agent modeling for interactive systems. In Benyon DR, Palanque P (eds). Critical issues in user interface systems engineering. Springer, Berlin, pp 191–210.

Davis AM (1993). Software requirements: objects, functions and states. Prentice-Hall, Englewood Cliffs, NJ.

DeMarco T (1979). Structured analysis, systems specification. Yourdon Press, Englewood Cliffs, NJ.

de Souza C (1993). The semiotic engineering of user interface languages. International Journal of Man–Machine Studies 39:753–773.

Dennett D (1989). The intentional stance. MIT Press: Cambridge, MA.

Diaper D (ed) (1989). Task analysis for human–computer interaction. Ellis Horwood, Chichester.

Diaper D, Addison M (1992). Task analysis and systems analysis for software development. Interacting with Computers 4(1):124–139.

Dowell J, Long J (1989). Towards a conception for an engineering discipline of human factors. Ergonomics 32(11):1513–1535.

Dowell J, Long J (1998). Conception of the cognitive engineering problem. Ergonomics 41(2):126–139.

Draper S (1993). The notion of task in HCI. In Ashlund S, Mullet K, Hendersen A, Hollnagel E, White T (eds). Proceedings of InterCHI'93: adjunct proceedings. ACM Press, New York.

Eco U (1976). A Theory of semiotics. Indiana University Press, Bloomington, IN.

Eco U (1984). Semiotics and the philosophy of language. Indiana University Press, Bloomington, IN.

Engström Y, Escalante V (1996). Mundane tool or object of affection? The rise and fall of the postal buddy. In Nardi B (ed). Context and consciousness: activity theory and human–computer interaction. MIT Press, Cambridge, MA, pp 325–374.

Ergonomics (1998). 41(2) 126–178.

Fischer G (1989). Human–computer interaction software: lessons learned, challenges ahead. IEEE Software (January):44–52.

Flach J (ed) (1995). Global perspectives on the ecology of human–machine systems. Erlbaum, Hillsdale, NJ.

Gibson JJ (1979). The ecological approach to visual perception. Houghton-Mifflin, Boston, MA.

Goel V, Piroli P (1992). The structure of design problem spaces. Cognitive Science 16:395–429.

Gray W, John B, Stuart R, Lawrence D, Atwood MA (1990). GOMS meets the phone company: analytic modeling applied to real-world problems. In Diaper D et al (eds). Human–computer interaction: INTERACT'90. Elsevier Science, Amsterdam.

Green TRG (1995). Looking through HCI. In Kirby M, Dix A, Finlay J (eds). People and computers X. Cambridge University Press, Cambridge, UK, pp 21–36.

Green TRG (1998). The conception of a conception. Ergonomics 41(2):143–146.

Green TRG, Benyon DR (1996). The skull beneath the skin: entity-relationship modeling of information artefacts. International Journal of Human Computer Studies 44(6):801–828.

Greenbaum J, Kyng M (eds) (1991). Design at work: cooperative design of computer systems. Erlbaum, Hillsdale, NJ.

Guarino N, Poli R (eds) (1995). Special issue: the role of formal ontology in the information technology. International Journal of Human Computer Studies 43(5/6):623–965.

Hix D, Hartson HR (1993). Developing user interfaces. Wiley, New York.

Hollan J, Hutchins E, Kirsh D (2000). Distributed cognition: toward a new foundation for human-computer interaction research.

Hollnagel E (1997). Building joint cognitive systems: a case of horses for courses? In Smith MJ, Salvendy G, Koubek R (eds). Design of computing systems: social and ergonomic considerations. Elsevier Science, Amsterdam, pp 39–42.

Hollnagel E, Woods DD (1983). Cognitive systems engineering: new wine in new bottles. International Journal of Man Machine Studies 18:583–600.

Hutchins E (1991). How a cockpit remembers its speeds. Manuscript. Department of Cognitive Science, University of California, La Jolla, CA. Cited in Nardi (1996).

Hutchins E (1995). Cognition in the wild. MIT Press, Cambridge, CA.

IJHCS (1997). Special issue: using explicit ontologies in knowledge-based systems development. International Journal of Human Computer Studies 46(2/3):181–408.

Imaz M, Benyon DR (1999). How stories capture interactions.

Jacobson I (1995). The use case construct in object-oriented software engineering. In Carroll JM (ed). Scenario-based design: envisioning work and technology in system development. Wiley, New York, pp 309–336.

Jacobson I, Chistensen, M, Johnson P, Overgaard G (1993). Object-oriented software engineering. Addison-Wesley, Reading, MA.

Johnson P, Johnson H, Wilson S (1995). Rapid prototyping of user interfaces driven by task models. In Carroll JM (ed.). Scenario-based design: envisioning work and technology in system development. Wiley, New York, pp 209–246.

Jones CC (1981). Design methods: seeds of human futures (2nd edn). McGraw-Hill, London.

Kangassalo H (1983). Structuring principles of conceptual schemas and conceptual models. In Bubenko J (ed). Information modeling. Chartwell-Bratt, pp 223–307.

Kao D, Archer NP (1997). Abstraction in conceptual model design. International Journal of Human–Computer Studies 46:125–150.

Kaptelinin V (1996). Activity theory: implications for human–computer interaction. In Nardi B (ed). Context and consciousness: activity theory and human–computer interaction. MIT Press, Cambridge, MA, pp 103–116.

Kaptelinin V, Nardi B, Macaulay C (1999). The activity checklist: a tool for representing the space of context. Interactions 6(4):27–39.

Karat J (1991). Taking software design seriously. Academic Press, New York.

Kay A (1990). A personal view. In Laurel B (ed). The art of human–computer interface design. Addison-Wesley, Reading, MA, pp 191–208

Kieras D, Polson PG (1985). An approach to the formal analysis of user complexity. International Journal of Man Machine Studies 22:365–394.

Kyng M, Mathiassen L (eds) (1997). Computers and design in context. MIT Press, London.

Lakoff G (1987). Women, fire and dangerous things: what categories reveal about the mind. Chicago University Press, Chicago, IL.

Lakoff G, Johnson M (1999). Philosophy in the flesh: the embodied mind and its challenge to western thought. Basic Books, New York.

Langefors B (1966). Theoretical analysis of information systems. Studentliteratur, Lund, Sweden.

Laszlo P (1969). System, structure and experience. Gordon & Breach, London, 1969.

Lefebvre H (1991). The production of space. Blackwell, Oxford.

Leont'ev AN (1978). Activity, consciousness and personality. Prentice-Hall, Englewood Cliffs, NJ.

Lewis P (1994). Information systems development. Pitman, London.

Lim, Long J (1994). The MUSE method for usability engineering. Cambridge University Press, Cambridge, UK.

Long J, Dowell J (1988). Cognitive engineering. Ergonomics 41(2):174–178.

Maclean A, Young RM, Belotti VME, Moran TP (1991). Questions, options and criteria: elements of design space analysis. Human–Computer Interaction 6:201–250.

Macaulay C, Benyon DR and Crerar A (2000). Etinography, theory and systems design: from institution to insight. International Journal of Human Computer Studies 53(1):35–60.

McCall R and Benyon DR (2002). Navigation: within and beyond the metaphor in interface design and evaluation, in Höök K, Benyon DR and Munro A (eds) Designing Information Spaces: The Social Navigation Approach. Springer-Verlag, London (in press).

Moran T (1981). Command language grammar: a representation for the user interface of interactive computer systems. International Journal of Man Machine Studies 15:3.

Nardi B (ed) (1996). Context and consciousness: activity theory and human–computer interaction. MIT Press: Cambridge, MA.

Nardi B (1997). HCI 97 keynote address. In BCS HCI'97 conference, Bristol, UK.

Norman D (1999). The invisible computer. Bradford Books.

Norman D, Draper S (1986). User-centred systems design. Erlbaum, Hillsdale, NJ.

Olle TW, Sol HG, Tully CJ (eds) (1983). Proceedings of IFIP WG8.1 working conference: comparative review of information systems design methodologies: a feature analysis. North-Holland, Amsterdam.

Olle TW, Sol HG, Verrijn-Stuart AA (eds) (1982). Proceedings of IFIP

WG8.1 working conference: comparative review of information systems design methodologies: a feature analysis. North-Holland, Amsterdam.

O'Neill E, Johnson P, Johnson H (1999). Representations and user-developer interaction in cooperative analysis and design. Human–Computer Interaction 14(1,2):43–92.

OVID (1997). Roberts D, Berry D, Isensee S (1997). Object view and interaction design. Tutorial notes, INTERACT'97.

Palanque P, Bastide R (1996). Task-models – system models: a formal bridge over the gap. In Benyon DR, Palanque P (1996). Critical issues in user interface systems engineering. Springer, Berlin, pp 65–80.

Parsons J, Wand Y (1997). Choosing classes in conceptual modeling. Communications of the ACM 40(6):63–69.

Payne SJ, Green TRG (1989). Task-action grammar: the model and its developments. In Diaper D (ed). Task analysis for human–computer interaction. Ellis Horwood, Chichester, pp 75–107.

Preece JJ, Rogers YR, Sharp H, Benyon DR, Holland S, Carey T (1994). Human–computer interaction. Addison-Wesley, Reading, MA.

Pylyshyn ZW (1984). Computation and cognition. MIT Press, Cambridge, MA.

Rasmussen J (1986). Information processing and human–machine interaction: an approach to cognitive engineering. Elsevier Science, New York.

Rasmussen J (1987). Cognitive engineering. In Bullinger H-J, Shackel B (eds). Human–computer interaction – INTERACT'87. North-Holland, Amsterdam, pp xxv–xxx.

Rasmussen J, Pejtersen A-M (1995). Virtual ecology of work. In Flach J (ed). Global perspectives on the ecology of human-machine systems. Erlbaum, Hillsdale, NJ, pp 121–156.

Redmond-Pyle D, Moore A (1995). Graphical user interface design and evaluation. Prentice-Hall, London.

Rosenquist CJ (1982). Entity-life cycle models and their applicability to information systems development life cycles. Computer Journal 25(3):307–315.

Rosson MB, Alpert SR (1990). The cognitive consequences of object-oriented design. Human Computer Interaction 5:345–379.

Rosson MB, Carroll JM (1995). Narrowing the specification–implementation gap in scenario-based design. In Carroll JM (ed). Scenario-based design: envisioning work and technology in system development. Wiley, New York, pp 247–278.

Rumbaugh J, Blaha M, Premerelani W, Eddy F, Lorensen W (1991). Object-oriented modeling and design. Prentice-Hall, Englewood Cliffs, NJ.

Schön DA (1983). The reflective practitioner: how professionals think in action. Basic Books, New York.

Shlaer S, Mellor SJ (1992). Object life cycles. Prentice-Hall, Englewood Cliffs, NJ.

Simon H (1981). The sciences of the artificial. MIT Press, Cambridge, MA.

Sloane A, van Rijn F (2000). Home informatics and telematics. Kluwer, Dordrecht.

Spence R (2000). A framework for navigation. International Journal of Human–Computer Studies 51:919–945.

Stamper R. (1977). Information. Batsford, London.

Storrs G (1989). Towards a theory of HCI. Behaviour and Information Technology 8(5):323–334.

Sturrock J (1986). Structuralism. Collins, London.

Suchman L (1987). Plans and situated actions: the problem of human–machine communication. Cambridge University Press, Cambridge, UK.

Suchman L (1995). Making work visible. Communications of the ACM 38(9):56–64.

Sully P (1994). Modeling the world with objects. Prentice-Hall, Englewood Cliffs, NJ.

Sundgren B (1975). The theory of database. Mason/Charter.

Sutcliffe A, Maiden N (1998). The domain theory for requirements engineering. IEEE Transactions on Software Engineering 24(3):174–196.

Sutcliffe A, Benyon DR, van Assche F (eds) (1996). Domain knowledge for interactive system design. Chapman & Hall, London.

Tweedie L (1995). Interactive visualisation artefacts: how abstractions inform design. In Kirby M, Dix A, Finlay J (eds). People and Computers X. Cambridge University Press, Cambridge, UK, pp 247–266.

Vicente K (1998). An evolutionary perspective on the growth of cognitive engineering: the Riso genotype. Ergonomics 41(2):156–159.

Vicente K (1999). Cognitive work analysis: toward safe, productive and healthy computer-based work. Erlbaum, Mahwah, NJ.

Vicente K, Rasmussen J (1992). Ecological interface design: theoretical foundations. IEEE Transactions on Systems, Man and Cybernetics 22(4):589–606.

Viller S, Sommerville I (1999). Coherence: an approach to representing ethnographic analyses in systems design. Human–Computer Interaction 14(1,2):9–40.

Winograd T, Flores F (1986). Understanding computers and cognition: a new foundation for design. Ablex, Norwood, NJ.

Wirfs-Brock R, Wilkerson B, Weiner L (1990). Designing object oriented software. Prentice-Hall, Englewood Cliffs, NJ.

Wittgenstein L (1953). Philosophical Investigations (3rd ed.) Trans. GEM, Anscombe, OUP.

Wood D (1992). The power of maps. Routledge, London.

Correspondence and offprint requests to: D. Benyon, School of Computing, 10 Colinton Road, Edinburgh EH10 1LD, UK. Email: d.benyon@dcs.napier.ac.uk