1

Understanding Task Analysis for Human-Computer Interaction

Dan Diaper
Bournemouth University

Task analysis is at the core of most work in human-computer interaction because it is concerned with the performance of work, and this is what, crucially distinguishes it from other approaches. A systemic approach is adopted to describe the properties of the models used in task analysis. The roles of task analysis in HCI and software engineering are introduced before practical issues, concerning how to perform the typical, early stages that are common to most task analysis methods, are addressed. The main example used throughout the chapter involves air traffic control and the theoretical and practical issues are finally illustrated within a method called systemic task analysis.

The goal of this chapter is to provide and illustrate the fundamental concepts that underlie task analysis so that its readers will subsequently find that task analysis is easy to understand.

1.1 TASK ANALYSIS IS EASY

The claim that "task analysis is potentially the most powerful method available to those working in HCI and has applications at all stages of system development, from early requirements specification through to final system evaluation," is Diaper's (1989a) conclusion. Although not disputing the potential of task analysis Anderson et al. (1990) suggested that "confusion reigns" and that "designers and human factors specialists fumble with the concept" of task analysis. If you agreed with Anderson et al. a dozen years ago, then, until you've read this chapter, you will probably still agree with them. The purpose of this chapter is to demonstrate that task analysis is easy to understand. This, of course, is a lie.

If you really want to understand the task analytic perspective on HCI, then one must have a reasonable competence in at least philosophy, psychology, sociology, ergonomics, and of course computing. On the other hand, although these disciplines may be part of the essential professorial armory, there is only a limited global need for theoretically driven academics, notwithstanding how necessary it is to have a few of them, so that most people interested in task analysis, and virtually all practitioners who want to use task analysis, do not need the arcane baggage of the theorizing academic. It is desirable, however, to recognize where such arcana fit into task analysis, and this chapter marks at least some places where some truly complex issues are relevant.

This chapter is unashamedly systemic, although this is not a novel approach. Shepherd (2001), for example, starts his book on hierarchical task analysis (HTA) by presenting a systems approach, and both the HTA and Goals, Operators, Methods, and Selection Rules (GOMS) methods are located within a systemic approach in the chapters of this handbook that describe them (chaps. 3 and 4, respectively). A particular style of systems model based on Checkland's (1981) soft systems methodology (SSM) conceptual models (Diaper, 2000a; Patching 1990), is used throughout the chapter in conjunction with Dowell and Long's (1989; Long, 1997) general HCI problem. Although a number of different examples are used in this chapter, the main one involves air traffic control (ATC).

1.1.1 Chapter Summary

If this chapter were a sentence, there would be a semicolon between sections 1.4 and 1.5. The first half of the chapter introduces the architecture of a systemic model (section 1.2); describes how this must be extended to model the performance of work, which is the core concept for task analysis, and explains why task analysis is at the heart of nearly all HCI activities (sec. 1.3); and then places task analysis in the context of HCI and software engineering (section 1.4). The second half of the chapter (section 1.5) addresses the use of task analysis in computing-related projects (section 1.5.1) and then introduces the common first stages of most task analyses, although some methods do use these stages in more or less an abbreviated form (section 1.5.2). A new task analysis method, systemic task analysis (STA), is introduced at the end of the chapter (section 1.5.3) to illustrate both the theoretical and practical issues discussed throughout.

Task analyses produce one or more models of the world, and such models describe the world and how work is performed in it (section 1.2). Descriptive models of the world consist of two types of entity: things and the relationships between things. Things can be physical or intangible, and most things are parts of other things (section 1.2.1). Intangible things include mental and software processes, emergent systemic properties, and models that are information rather than physically based. Task analysis models are among the intangible things.

Relationships between things in models are of two sorts: conceptual and communicative (section 1.2.2). Conceptual relationships (section 1.2.2.1) typically concern classifying things. Many taxonomic systems, including those used in task analysis, use a hierarchical structure, although, it is argued, much of the natural world is not truly hierarchic and would be better modeled as a heterarchy, using a "level of abstraction" concept while allowing things to have multiple memberships in other things. Communicative relationships in descriptive models of the world (section 1.2.2.2) involve how different things affect each other. Adding these relationships to taxonomic ones reinforces the suggested advantages of heterarchies over hierarchies and allows the introduction of the basic systemic model, based on SSM, used throughout the rest of the chapter.

What makes task analysis in HCI distinctive is its primary concern with the performance of systems (section 1.3.1), and performance in task analysis is fundamentally about doing work to achieve goals (section 1.3.2). (Dowell and Long's 1989; Long 1997) general HCI problem distinguishes between the work system and its application domain. A work system in HCI typically is made of people, including users, and computer systems and usually other things as well. Work is performed by the work system so as to change its application domain, and it is successful if the changes meet the goals of the work performed. Complex systems will contain numerous overlapping work systems and application domains (section 1.3.3), and it is necessary for task analysts to appropriately identify the work system and application domain of interest.

This chapter defines HCI as an interdisciplinary engineering discipline that is a subdiscipline of ergonomics (section 1.4). Furthermore, it suggests that the historical division between HCI and software engineering is unfortunate, as both study the same sort of systems for similar engineering purposes. It then introduces two views of HCI, one broad and the other narrow (section 1.4.1), and relates these views to different definitions of work systems and application domains. Task analysis needs to have models of people's psychology within the systems that are studied, and the chapter therefore discusses various styles and types of cognitive model (section 1.4.1.1). Task analysis must also consider the psychology of those who are involved in computing-related engineering projects, as they are either the final end users of task analysis methods or users of the results from task analyses (section 1.4.1.2).

Software engineering's solutions to the software crisis have been primarily anthropocentric because the crisis is one of human productivity (section 1.4.2). Its solutions have been to develop methods for working on software projects and to support many of these methods with computer-assisted software engineering (CASE) tools. The roles of task analyses need to be located within the software life cycle. Within projects, task analyses vary in their fidelity (i.e., how well they relate to the assumed real world) and are best thought of as always studying simulations or selected examples of their world of interest (section 1.4.2.1). The chapter emphasizes the importance of the concept of agent functionality and discusses the allocation of function between different things in a work system (section 1.4.2.2). It also points out the need for task analytic representations to be device independent in the context of abstracting task representations and discusses the possible role of goals in such abstraction (section 1.4.2.3).

The second half of the chapter starts by arguing that task analysis can be of use in most stages of any computing project (section 1.5). Stressing that any practical method must involve considerable iteration (section 1.5.1.1), the chapter suggests five questions that should precede undertaking a task analysis:

1. Which project stages will use task analysis? (section 1.5.1 and 1.5.1.1)
2. What knowledge is the task analysis intended to provide? (section 1.5.1.2)
3. What is the desirable task analysis output format? (section 1.5.1.3)
4. What data can be collected? (section 1.5.1.4)
5. What task analysis method? (section 1.5.1.5)

Following discussion of these five questions, the chapter describes the firsts two stages of most task analysis methods: (a) data collection and representation (section 1.5.2.1) and (b) activity list construction and the classification of things (section 1.5.2.2). It is after these stages that different task analysis methods diverge (section 1.5.2.3).

To illustrate these early common stages, it is necessary to choose a method and notation. Systemic task analysis (STA) is used because it has been developed from the ideas exposed in the first half of the chapter (section 1.5.3). In describing its first stage (section 1.5.3.1), the chapter uses the ATC example previously introduced but adds more project-specific details. The chapter describes how video data might be collected in an ATC tower (section 1.5.3.2), constructs a sample fragment of an activity list from these data, and proposes that sometimes such construction is sufficient task analysis in some projects or project stages (section 1.5.3.3). Although STA uses a formal, heterarchical modeling method, the chapter offers the start of a hierarchical analysis of the fragment of the activity list (section 1.5.3.4) to illustrate how task analysis methods might proceed following the fairly common early stages.

1.2 DESCRIBING THE WORLD

Does the world exist? The philosophical tradition of solipsism denies that this question can be answered, and even Descartes' proof of his own existence, encapsulated in his famous phrase "Cogito ergo sum" (I think, therefore I am), has for many years been recognized as a conclusion based on a faulty argument (Watling, 1964). Science and engineering, however, traditionally require belief in a real world external of the mind. The position of the author, which he calls *pragmatic solipsism*, is an intermediate one. It treats the world's existence as an assumption, and one corollary is that all that can ever be known about the world are merely models. Such models of the world can be compared, and some will be preferred over others for some purposes. In science, for example, Ockham's razor might be applied to choose the simplest of two equally good competing theories, although this begs questions of what is meant by "simplest" and "good." This chapter is based on the pragmatic solipsist assumption and argues that one useful class of models of the world for engineering purposes consists of the task analytic ones. It then attempts to elucidate the common nature of this class of model while recognizing that no model of the world can be proved to be true. It thus admits alternative models, which might be called *perspectives* (see also, e.g., chap. 7), and allows that some will be better suited for some purposes than others.

At the core of any task analytic model there are two things:

1. A description of the world.
2. An account of how work is performed in the described world.

It is the latter of these that differentiates task analysis from approaches that are primarily descriptive. These two are sometimes combined, but it is almost certainly easier to maintain a conceptual separation between them, as this chapter does.

A model is a description of the assumed real world. A painting or a poem can be a model of the world, as can the sorts of diagrammatic, logical, or language-based models used in HCI and software engineering. Design requires two models of the world: a current one and a future one. Design is a goal-directed activity involving deliberate changes intended to improve the current world, so the need to model the future in design is unquestionable. In practice, models of possible future worlds need to be based on models of the current world because the world is a very complicated place and accurately predicting the future must accommodate its complexity (see also chap. 30). In addition, constructing a model of the current world is easier because the current world can be observed and because the model can often be scientifically tested to determine if what the model predicts to be the case is actually so. For example, an analyst might construct a model of the current world that predicts that on web pages animated images surrounding text will interfere with the task of searching the text for information, as Zhang (1999) proposed. This prediction could be tested using typical realistic web pages, as Diaper and Waelend (2000) did. They found that the prediction was false. As Alavi (1993) stated, "Most research studies have been conducted in laboratory settings that limit the generalizability of their findings to more complex organizational settings."

In contrast to laboratory-based science, task analysis creates its models of the current world by studying it in situ. Famously, it involves the observation of people performing their tasks in an environment as realistic as possible. The performance may be documented using video cameras or by taking notes. Task analysis can also make use of other data collection methods (section 1.5.1.4; Diaper, 1989a) such as interviews (e.g., chap. 16), immersive methods such as those used in ethnography (see chap. 6 and 14), or various forms of invented scenarios (see chap. 5).

Unfortunately, since descriptions of the world in HCI include people in complex social and organizational environments, truly accurate prediction of any postdesign future is virtually impossible, not least because people often (a) adapt their behavior to designed changes, (b) alter other aspects of the world to accommodate what has been designed, and (c) change and use what has been designed in ways unanticipated by the designers. As Sharples (1993) commented, "The way designers intend technology to be used very often differs from the actual users' behavior" (p. 67).

Prototyping approaches (section 1.4.2) offer only a partial solution to the problem of predicting postdesign worlds (Carroll, 1991), and the underlying rationale for providing the best possible model of the current world is firstly one of cost, as the fewer cycles of prototyping required to achieve a satisfactory design, the cheaper will be the overall design exercise. Perhaps even more importantly, design is less likely to fail the more accurately the future world can be predicted, and a model of the future will almost certainly be more accurate if the model of the current world is accurate.

Descriptive models of the world tend to be constructed from two general classes of entity:

1. Things.
2. Relationships.

The next two subsections introduce these basic components. The discussion ignores time and even sequence and therefore the performance of work, which is introduced in the next main section (1.3).

1.2.1 Things

A model represents the world as being made of something, variously called things, objects, entities, items, components, pieces, parts, attributes, properties, and so forth. In HCI the sorts of things that are used in its models include people, computer systems. and usually other things as well. Things can be parts of other things, so that a user-computer interface might be considered to consist of things called computer output and input devices, such as a screen and loudspeakers and a keyboard and mouse.

Although it is reasonable from a systemic perspective to treat people as just another sort of thing, they have a special status—and not just because they are far more complicated than anything else, to the point that they do not even understand themselves (Diaper, 1989b, 2002a). People individually possess moral attributes, and cannot be treated like other things. For example, unlike computers, they cannot be legally turned off (killed) or intentionally damaged. In addition, people as individuals must ultimately be considered as moral atoms, even though moral issues often involve two or more people. For example, we might consider a particular political system to be immoral, but our reason for viewing it as immoral is likely to be its harmful effects on some people.

Most models of the world contain some things that have an assumed tangible, physical reality (i.e., they can be perceived by the five human senses), and usually such things provide a starting point for model building. Many things in HCI models, however, are intangible, most obviously things that are mental or are software. Similarly, the concept of intangible properties is central to systemic models because, as Patching (1990) stated, "It is not possible to see, hear or touch a social, or political, or industrial relations system" (pp. 9–10). Of course, this leads to some fundamental philosophical problems that while of interest to the theorizing academic, are not germane to the practice of task analysis. From a pragmatic point of view, what is important in them). Analysts' models of the world are one example that fall into this category of intangible thing. Similarly, software can also believe things about the world, although the tendency is to blame the programmers. For example, prior to the 2K revisions, many programs believed that "no person is older than 100 years" and were automatically going to assign the age of 1 year to people born in 1899 upon the arrival of the new millennium.

Many of the things used in HCI and task analysis are described as information rather than physically. Although, for convenience, it is common for physical things to be described as containing information, care needs taking with the concept of information, as this is not really a property of a physical thing but of some other thing, usually an agent, that perceives the physical object. This is obvious when one considers the different information supposedly contained in a document perceived by a person who understands the language in which the document is written and by a person who does not. Apart from physical and informational descriptions, things can be described in other ways, depending on the perspective chosen by the analyst (e.g., money, power, or responsibility). These alternatives, however, tend to be used in only a small percentage of task analyses. That still leaves a wide range of different perspectives to choose from. This chapter assumes that psychological, sociological, and other such perspectives are basically information-processing ones, even when their focus is on, for example, affective issues (see chap. 29), because affect changes a person's view of the world, as Nussbaum (2001) argues. Section 1.5.3.3 presents an example in which the psychological property of trust between coworkers is important. Chapter 14 describes other sorts of relevant psychological properties, such as fatigue effects, and chapter 28 reviews psychological theories and models in detail. Although Checkland's SSM is celebrated for its multiple-perspectives modeling of the world, the integration of different perspectives remains a problem that, except in some specialized cases such as physical reductionism (Diaper, 1984), is epistemologically intractable and only solvable by the application of the analyst's craft skills (Long, 1986; Long & Dowell, 1989).

Just what is a thing is a complex metaphysical question (e.g., Lowe, 2002). Ignoring metaphysics, however, in practice task analysis traditionally divides its things into two main types, objects and actions (Diaper, 1989c). Often objects are further categorized as agents or things used by agents. Actions, of course, are what agents do, alone or with their objects and other agents. An activity list line, which can be used to describe a task step (as recommended in section 1.5.2.2), typically has a single agent, a main action and perhaps some subordinate ones, and the recipients of the action (i.e., objects and other agents). People generally seem to understand what is meant by an agent, and some methods in the task analysis literature describe actions and objects as verbs and nouns, respectively, although this is only an approximate metaphor. Still ignoring the truly complex metaphysical issues, the key property of agents is their ability to initiate behavior whereas other objects are passive (see also chap. 26). For example, a person putting words down on paper is an agent performing the action of writing, and the passive objects are the pen, ink, and paper. If the paper is given to someone else to read, then its transmission is another task, initiated by one agent and with another as the recipient, even though some properties of the paper (e.g., its location) have also changed. Probably the real difference between agents and other objects is that the latter have processes that are adequately describable by the physical sciences (physics, chemistry, some biology, etc.) whereas agents have processes that, because they involve emergent properties (section 1.2.1) or cognitive or computer information processing (e.g., chap. 15), require other sciences to be brought in. One conceptual difference between using a word processor rather than pen and paper is that with the word processor the writing is to another agent—the computer system. Thus, dialogue-based models are common in computing applications whereas they are not used for objects describable by the physical sciences alone. Other sorts of classifications of things are possible (e.g., some things can be classified as triggers; see chaps. 14 and 19).

1.2.2 Relationships

All things in a model have at least one relationship to another thing and often more than one relationship with other things. Isolates that have no interaction with anything can have no effect on the rest of the system and so can be legitimately ignored.

Although the following distinction may not be absolutely true theoretically, it is useful to classify relationships between things in models as of two types, conceptual and communicative.

1.2.2.1 Conceptual Relationships

Conceptual relationships are typically taxonomic (see chap. 20 for a discussion of taxonomies and classification schemes). Probably the simplest type of conceptual relationship concerns what things are made of, and these are typically modeled hierarchically (Diaper, 1984). Thus the user interface mentioned in section 1.2.1 is_made_of a computer output and input system that is_made_of the screen and loudspeakers (output) and the keyboard and mouse (input). Similarly, the computer system is_made_of the user interface and other unspecified computer hardware and software things and the complete system includes a computer and a user and the environment in which the user and computer system operate. Figure 1.1 illustrates this hierarchy using a tree diagram and a Venn diagram, which is a graphical representation of a formal, logical model based on set theory, the foundation of all modern mathematics (Ross & Wright, 1988). The two representations are logically identical (Diaper, 2001a); that is, they contain exactly the same information but in a different graphical format.

Other types of relationship may be used for taxonomic purposes, for example, kinship or the "is_similar_to" relationship. The latter can be a complex relationship if things are judged to vary on more than one attribute (an attribute is a property of a thing as well as a thing itself). Bowker and Star (2000) present a compelling case that taxonomic systems are common and important in most areas of human endeavour. Furthermore, not only do classification systems get used alone, they also form the basis for more complex types of models of the world, including task analytic models.

Conceptual relationships do not have to be specified hierarchically, although this is common in many HCI and software engineering methods and justified on the basis that people find hierarchies naturally easy to understand (see de Marco, 1979, on the design of data flow diagrams (DFDs)); Paterno, in chapter 24, claims that people's understanding of hierarchies is "intuitive." Diaper (2000a, 2001a, 2001b) has suggested the neologism *heterarchy* to describe models in which a thing is related to more than one other thing (see Fig. 29.1 for a simple example). In a tree-style model, such as Fig. 1a, heterarchical conceptual relationships can be represented either by repeating nodes in the tree or by allowing child nodes to have more than one parent node. This is not, however, an elegant solution, as is discussed below.

Although most things engineered by people do have a hierarchical design, it is far less clear that the natural world and our social worlds are arranged hierarchically (Diaper, 2000a). Biologists, for example, have found many an organism that is "*incertae sedis* meaning that its position in the classificatory system is uncertain" (Clark & Panchen, 1971). Diaper (2001a) demonstrated that even a simple office system's organization of people's roles needs to be modeled heterarchically when roles are shared. Thus, although hierarchically arranged DFDs, for example, may be appropriate if used to describe some types of software, it is of concern that methods such as HTA and many of the task analysis methods classified in chapters 6, 22, 24, and 25 may be forcing an inappropriate hierarchical organization on to their model of the world. In contrast, from its inception in the early 1980s, task analysis for knowledge descriptions (TAKD; Diaper, 1989c; Diaper & Johnson, 1989), used a heterarchical representation of a tree with nodal repetition, although it was mislabeled a hierarchy for a long time by its developers (Diaper, 2001b).

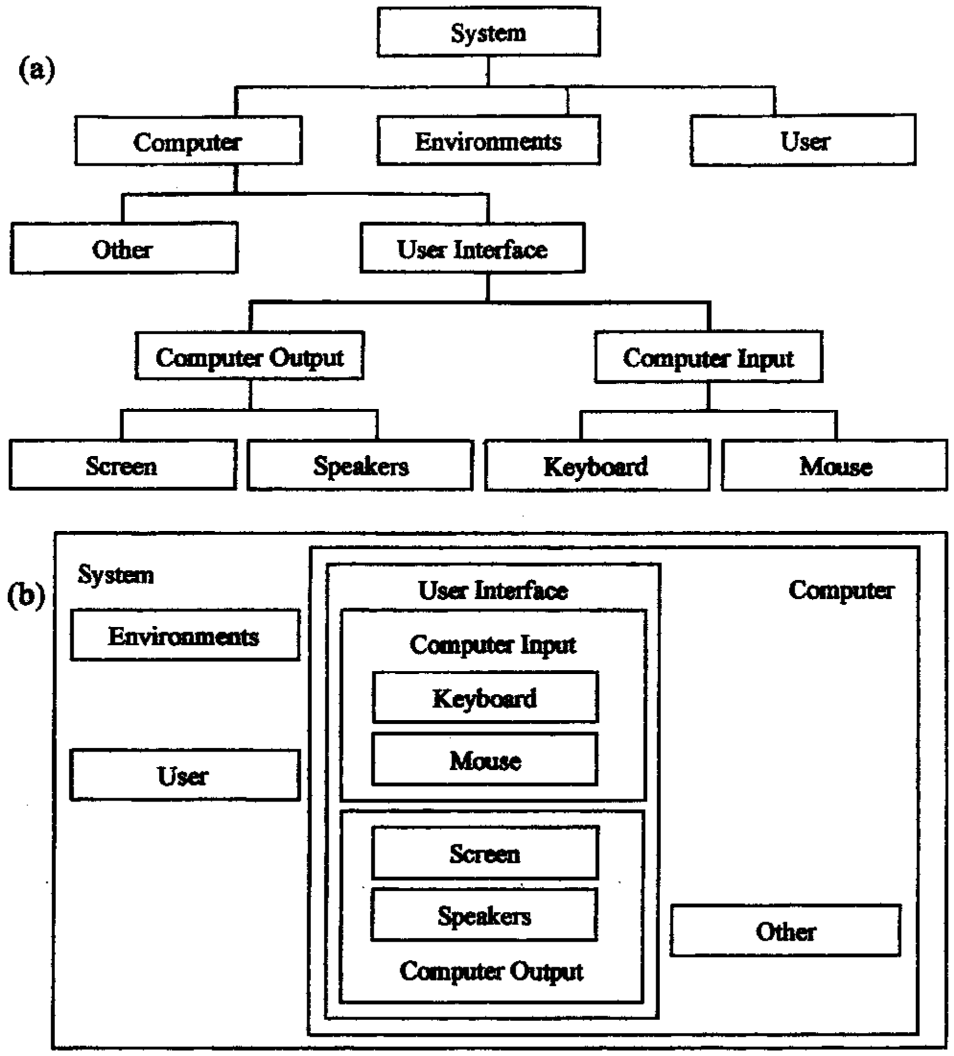

FIG. 1.1. Both 1(a) and 1(b) show the logically identical conceptual relationships between some components of a system that involves a computer and a user. Figure 1(a) represents this as a tree diagram and 1(b) as a Venn diagram. Note that the size and shape of the sets represented in a Venn diagram are not meaningful.

Given the familiarity that most people have with hierarchical models, then the currect practical advice is to start with a hierarchical structure but, when confronted with something *incertae sedis*, to consider moving to a heterarchical model. Using hierarchical models does have the advantage of forcing analysts to determine whether some thing is of one sort or another and thus encourages careful thought about things. There are times, however, when classification decisions have to be contrived, and heterarchicies provide a logical alternative that need not compromise a model's validity for the sake of maintaining a hierarchical structure.

1.2.2.2 Communicative Relationships

In a communicative relationship, two or more separate things communicate with or affect one another is some way. Communicative interactions require another type of thing, as all interactions between things are mediated by something. For instance, light from a computer's screen is the physical thing that links the screen to the user's eyes, and mechanical energy is

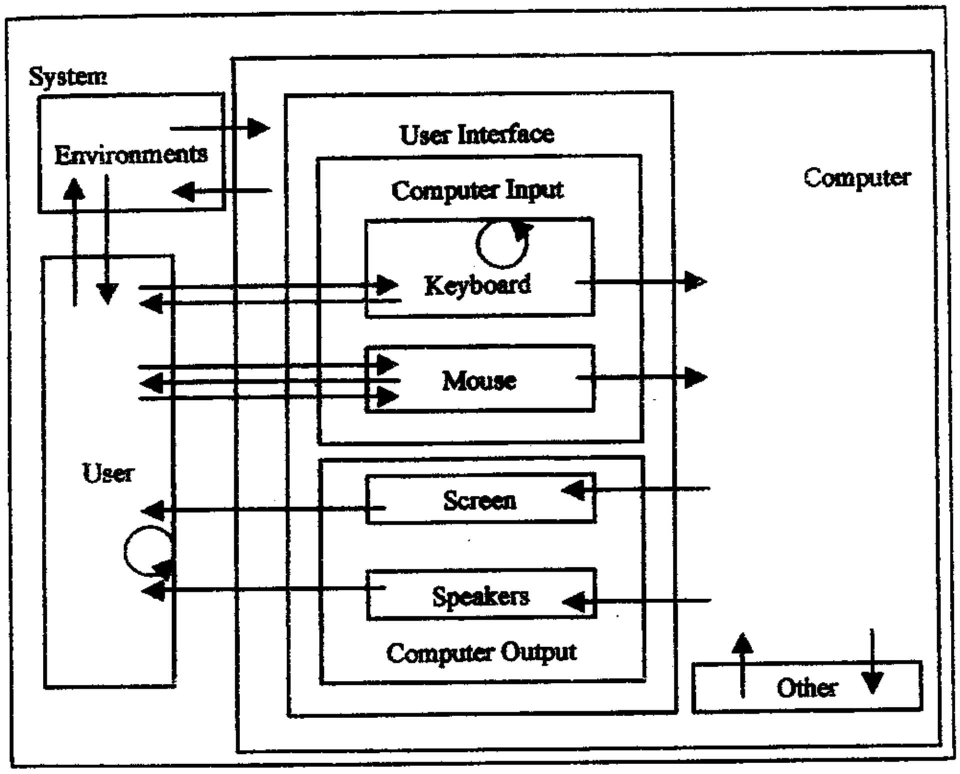

FIG. 1.2. A Venn diagram model of a system consisting of a user and a computer. The diagram shows the communicative relationships between the things in the model.

transferred between fingers and keys on a keyboard. In HCI, it is usual but not universal to represent communicative things (mediators of communicative relationships) as information that is being transmitted.

There are clear advantages to using a Venn diagram (Fig. 1.1.b) for modeling purposes rather than a tree diagram (Fig. 1.1.a). Among the pluses, communicative things can be added easily to the taxonomic structure using a quite different graphical notation to that representing conceptual relationships (see Fig. 1.2). In Fig. 1.2, the communicative things are represented by unlabeled arrows that overlay the hierarchical representation of Fig. 1.1.b. Thus, Fig. 1.2 is no longer a hierarchy but a heterarchy. That is, the communicative relationships can be within or between hierarchic levels, although only those that jump levels are shown in Fig. 1.2.

Not to label and describe arrows is too common a sin in HCI and software engineering modeling activities. The problem is that arrows often represent different types of thing. Diaper (2000a) argued that if defining arrows is a necessary requirement for simpler, single-perspective models such as DFDs, it is even more necessary for multimedia and multiple-perspective models. Illustrating such semantic differences, in Fig. 1.2 the input devices provide immediate kinaesthetic feedback to the user as they are touched or moved. The mouse is modeled as a two-function input device (movement is separated from button clicking), but whether there are two types or only a single type of feedback may depend on many things, including the nature of the task and the expertise of the user. The computer does not provide feedback to the input devices (i.e., the devices do not "know" whether they are connected to the computer or not). The keyboard's Caps Lock and Num Lock keys are represented by a circular arrow to indicate that they affect what the keyboard sends to the computer but do not themselves send information to the computer. Figure 1.2 does not show a relationship from the user to the computer output devices. The rationale for this is that, although the user may engage in attention-directing, orientating behaviors toward the output devices, nothing is communicated to the devices (i.e., a screen or loudspeaker does not know whether anyone is attending to its output). In many cases it is important not to forget these user behaviors and they can also be indicated, as in Fig. 1.2, by a circular arrow. The circular arrow might represent behaviors, such as head and eye movements, that are potentially observable by a task analyst as well as human attentional processes (see chap. 15) that are psychological and can only be inferred.

Users do not interact only with their computer systems but inhabit a rich, complicated environment made of other things, often including other people. The same is true of computer systems. Fig. 1.2 has not decomposed the environment of either the user or the computer system. It should be noted, however, that some aspects of the "environments" may be shared by both user and computer and that some, usually most, will be unique to just one of these. If a decomposition using a hierarchical representation is attempted, it will fail or at best be clumsy because of the shared things in the environments. Heterarchical models, which allow things to be components of more than one higher level thing, avoid this problem.

Figure 1.2 is potentially a systems model. It follows the conventions of SSM conceptual models but is drawn to stricter criteria, based on DFDs, than is common in SSM (Diaper, 2000a). It is, however, a static, atemporal model. That is, it describes the world independently of time or even sequence and therefore independently of events, because events must occur in time. It is common for analysts to run such a static model in their minds similar to the way programmers run the code they are writing in their minds (in both cases to predict run-time performance). This is doing an implicit task analysis because it is modeling system performance. Diaper, McKearney, & Hurne, (1998) argued that in realistically complicated systems an implicit task analysis is always likely to be done poorly, and they presented their Pentanalysis Technique as a method for turning implicit task analyses into explicit ones.

1.3 PERFORMANCE AND WORK

Task analysis focuses on the performance of systems. Annett and Stanton (1998; see also Diaper, 2002b, 2002c) suggest this when they state that task analysis is the collective noun used in the field of ergonomics, which includes HCI, for all the "methods of collecting, classifying and interpreting data on human performance" (p. 1529). Perhaps because both Annett and Stanton have a psychology background, they emphasize "human performance." From a systems perspective, it would be preferable to alter the definition to read thus:

Task analysis is the collective noun used in the field of ergonomics, which includes HCI, for all the methods of collecting, classifying, and interpreting data on the performance of systems that include at least one person as a system component.

The two key concepts in this definition are performance and data. Performance is how a system behaves over time. A datum is the description of some part of a system at a point in time, and data may be collected over time and from different parts of the system.

1.3.1 Performance

The performance of a system is how it behaves over time. Systemic time can be characterized as a series of instants each representing the system in a particular state. Systems that have human and computer components have many more instants than can possibly be observed, so what is available to analysts is a series of snapshots of the state of the system with missing states between these, like the frames of a cinema film. In practice, what is available to analysts as data is the representation of the state of only one or a very small number of system components. To extend the cinema film analogy, the analyst cannot see all of any frame but only one or a few areas of it. Thus, all but the most trivial systems will appear to operate in parallel. That is, at each instant more than one component of the system can be expected to have changed in some way, and the analyst cannot know the states of all the components that do change from instant to instant. The problem is the same one that bedevils testing any large software system but worse, that a system state is possible that prevents the expected operation of a part of the system being observed and so making it difficult to identify the cause or causes of the unexpected system state.

Figure 1.2 can be used to illustrate this. The user can be observed pressing keys on the keyboard and we could test the ASCII string that is sent from the keyboard to the computer and so on until each character is echoed on the screen. Usually, as far as the user is concerned, this echoing loop from keyboard to screen is instantaneous, but sometimes it is perceptibly slow, sometime frustratingly so, for example, when the other computer components place a high demand on computer-processing resources.

One obvious question for user interface designers in the above scenario is, How slow can echoing be, in particular situations, without causing the user some sort of problem? This type of question is at the very heart of task analysis; as Annett, in (chap. 3), proclaims of HTA, the focus should be on solving problems. The issue is how to define "satisfactory performance" for a system and its components. The concept of work is critical to defining "satisfactory" in this context.

1.3.2 Work

Dowell and Long (1989; Long, 1997) have produced perhaps the most sophisticated framework for the discipline of HCI. Their general HCI design problem can be characterized at a high level as having two system components—a work system and an application domain—in an environment. Figure 1.3 illustrates the basic model, which can be described so:

Work is achieved by the work system making changes to the application domain. The application domain is that part of the assumed real world that is relevant to the functioning of the work system. A work system in HCI consists of one or more human and computer components and usually many other sorts of thing as well. Tasks are the means by which the work system changes the application domain. Goals are desired future states of the application domain that the work system should achieve by the tasks it carries out. The work system's performance is deemed satisfactory as long as it continues to achieve its goals in the application domain. Task analysis is the study of how work is achieved by tasks.

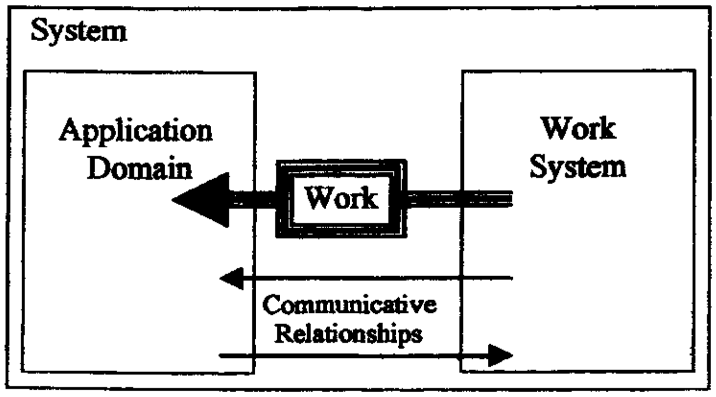

FIG. 1.3. The general model of work, based on Dowell and Long (1989; Long, 1997).

Dowell and Long's conception is that tasks are not performed within the work system but by the system as it acts on the application domain. Furthermore, the communicative relationships between the work system and application domain may be what changes the application domain, but they are not the same as the work performed, because work is defined in terms of the achievement of goals. Consequently, describing the communicative relationships is not sufficient for describing a task. Annett (chap. 3) comes to a similar conclusion when he states that "simply to list actions without understanding what they are for can be misleading." What is needed for task analysis is an understanding of how the communicative relationships arise in the work system and how they cause changes within the application domain—and, of course, vice versa (i.e., how feedback from the application domain arises and then affects the work system).

Throughout this handbook, the concept of goals is closely tied to that of tasks, and goals are nearly always described as being psychological, in that it is people who have them. In contrast, in the systemic version of Dowell and Long it is the work system that has goals, which specify the intended changes in the application domain. The advantage of this view of goals is that it recognizes that nonhuman things can have goals. In SSM, nonhuman goals reflect one aspect of what are called "emergent properties" of a system. Organizational systems, for example, can have goals that may be different from those of any individual person within the system. This is not a controversial claim in systemic approaches such as SSM or in any approach that uses an organic metaphor for organizational life. Certainly very few, if any, employees of a large business have a personal goal such as "increase market share," although this might well be a good partial explanation of the business's behavior with respect to its external environment (i.e., its application domain). Furthermore, this view of goals allows parts of an organization and individuals within it to have different goals and even ones that conflict. Indeed, a great deal of collaborative work is focused, not on achieving common goals, but on resolving conflict (Easterbrook, 1993; Shapiro & Traunmuller, 1993). Similarly, this view of goals makes it reasonable to suggest that things like computer systems can have goals separate from those of their users, a point explicitly made in chapter 26. Thus a business computer might have as a goal "to maintain client database accuracy" and attempt to achieve this by forcing users to operate in a way that they resist, perhaps because it requires more effort from them than they wish to expend. Although it has been suggested that goals in a computer system are simply those of its designers (Bench-Capon & McEnery, 1989a, 1989b), the alternative is that large computer systems possess properties, perhaps emergent ones, unanticipated by their many designers (Barlow, Rada, & Diaper, 1989; see also sec. 1.4.1.2).

It is probable that one of the main confusions that has arisen in task analysis over the years has been the assignment of different types of goals to individual people rather than to work systems. One goal of a data entry clerk, for example, might be "to achieve all the data entry work assigned as quickly as possible with a detectable error rate acceptable to management so as to obtain a bonus payment." In this case, the computer system (and perhaps the management checking function) is the clerk's application domain, and the clerk, the documents containing the data and other things, and the management, office, and social environments constitute the work system. Confusion arises, however, if the clerk is also taken to have the organizational goal "to increase market share." This is clearly not the case (if it were, bonus systems would not be necessary). Note that this argument is based on the psychological assumption that possession of a goal provides motivation, or willingness to act to achieve the goal (for further discussion, see chap. 15).

Teleology is that branch of philosophy concerned with causes, reasons, purposes, and goals. Goals are included because they are future desired states that require the same causal model. In the case of a goal, its achievement, which will occur in the future, can be caused by events happening now. Task analysis is generally mono-teleological in that behavior, particularly that defined through low-level, detailed descriptions, of behavior, is commonly represented as being caused by a single goal. Most of the chapters in this handbook promote a mono-teleological philosophy. A hierarchy of goals, as used in HTA, consists of multiple related goals, but a person can also perform an action on the basis of unrelated goals. Furthermore, unrelated goals that nonetheless motivate the same behavior cannot be simply prioritized in a list, because different goals have more or less motivational potency depending on their specific context.

For example, a chemical plant operator's unrelated goals for closing a valve might be (1) to stop the vat temperature rising, (2) to earn a salary, and (3) to avoid criticism from the plant manager. The first might concern the safety of large numbers of people, the second is sociopsychological and might concern the operator's family responsibilities, and the third is personal and might concern the operator's self-esteem. These three goals correspond to different analysis perspectives, the sociological, the sociopsychological, and the personal psychological; and there are other possible perspectives as well. Furthermore, people might have different goals within a single perspective.

Of course, the principle that any given behavior is likely to be caused by multiple goals can be extended to any complex work system, as discussed above. At the least, task analysts should be cautious whenever the explanation of the behavior of a work system is ascribed to a single goal, because there will probably be many, within and across different perspectives, and a set of goals will be prioritized differently in different contexts. Nor is a first-past-the-post threshold model likely to be satisfactory, as a combination of goals, rather than a single one, will most often reach some notional threshold and so trigger behavior. In the chemical plant operator's case, the usual main reasons to close the valve might be to avoid management criticism and ensure receiving a salary, but after an argument with the manager about salary, the operator's main goal might become protection of the health of people at risk from a chemical accident. If the latter now more strongly drives the operator than the other two goals, we might want to be quite sure that the operator is aware of the safety issues, which he or she might not be if the main motivating goals had previously been to avoid management criticism and ensure a salary.

Figure 1.4 provides a concrete but highly simplified example in traditional air traffic control (ATC). Chaps. 13 and 25 describe more modern computerized ATC systems. The application domain contains things like aircraft, weather, and so forth. The goal of ATC is to facilitate the safe and expeditious movement of aircraft, and this is both an organizational goal and one held by many of the workers. It is doubtful, however, whether it is one held by the canteen staff, for example, even though they could affect the accomplishment of the organizational goal if they fail to achieve their own goals, for examples concerning food hygiene. The simplified ATC work system in Fig. 1.4 consists of various people, the flight strip system, radar, radio and normal telephones, and so forth. The ATC work system carries out tasks intended to meet the ATC work system goals of the safe and expeditious movement of aircraft in the application domain. The concept of a goal is further explored in chap. 30, which takes a more radical position on the concept than that presented here.

One important representation of each aircraft in an ATC work system is the flight strip, which records the status of each aircraft and is updated by the ATC officer (ATCO) or the flight chief, who between them have responsibility for the aircraft represented by the flight strips in front of them. Sample flight strips are shown in chapter 19. Updating a flight strip is not a task of the ATC work system. It's not an ATC work system task because its performance does not directly change the application domain; that is, updating a flight strip does not change the real world of aircraft. A particular flight strip can only be correct or incorrect, satisfactory or not, with respect to other representations of the information within the ATC work system. The flight strip has no direct connection with its aircraft. It is conceptually related to the aircraft but not communicatively, except via other things. (Note that the conceptual relationship is not shown on Fig. 1.4, but it could be if this was the analyst's focus). If the ATCO updates a flight strip

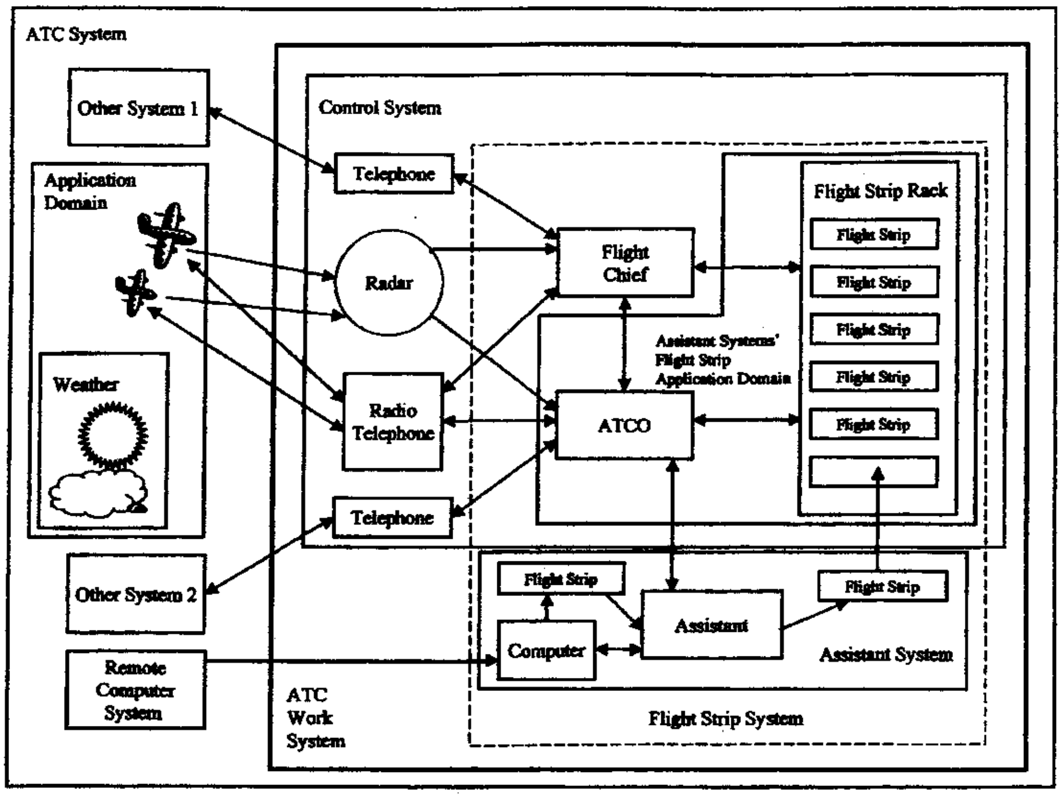

FIG. 1.4. A simplified model of a traditional air traffic control (ATC) system that uses paper-based flight strips.

with information that is incorrect with respect to the aircraft that the strip purports to represent but is correct with respect to the source of the information, say, incorrectly displayed aircraft information on a radar screen, then what is incorrect in the system is not the flight strip but the radar. In safety critical applications such as ATC, there is usually considerable deliberate information redundancy so that, on the principle of double-entry bookkeeping, information mismatches have a high probability of being detected. Provided a problem does not escape from within the ATC work system and that there are no time or other types of penalties incurred, then it has no effect on the satisfactory performance of work achieved in the application domain of aircraft movements.

The above example, in which updating a flight strip is shown not to be an ATC work system task, would appear to present a conundrum, as obviously, if a flight strip is not updated or is changed incorrectly, the result could be a real world "incident," what used to be called a "near miss" between aircraft in the application domain. The solution to this apparent problem with the Dowell and Long (1989) definition of work involves establishing what to compare the state of a flight strip with and using a multiple systems model that always has a work system and application domain for each of its embedded models.

1.3.3 Multiple Models

The final piece in this part of the jigsaw puzzle is the concept of multiple work systems and application domains. In theory, any set of components that have some communicative relationship, directly or by other things, can be designated by the analyst as constituting either a work system or an application domain. In practice, analysts will tend to assume that most of the possible work systems and application domains are not sensible. The minimal criterion to apply, however, is whether a definition of work is possible.

There are a couple of little theoretical twists that may help in understanding the ideas but do not usually have great practical relevance. First, with complex application domains, it is possible to consider reversing the work system and application domain. Although this would be insane in normal ATC, as it would mean that the purpose of the aircraft movements was to satisfactorily perform work by changing the ATC work system, it is a conceivable method of testing a new ATC system using real aircraft. Equally insane, at least superficially, is to suggest that a simple tool like a hammer performs work by changing its user, although the idea that people adapt to their tools does imply that using a hammer has effects on the hammerer, and this could be the goal of a tool's designer. Educational systems, of course, do explicitly have user-changing goals. The cynical view that real people are driven by bureaucracies or by computer systems rather than the reverse can be easily understood by switching application domain and work system. This concept is most useful when there are chains of work systems and application domains, but care needs to be taken, as often the application domain of work system$_n$ is not quite the same as that for work system$_{n+1}$.

The second twist is that, although many analysis methods use a hierarchical structure, it is not necessary to do so, and in many cases it is more valid to use heterarchical models (section 1.2.2). A heterarchy will greatly increase the number of work systems and application domains that can be considered for analysis. In practice, it is therefore necessary to be careful in defining both the application domain and the work system of interest.

Returning to the ATC example, the ATC work system is modeled as containing three of many possible work systems, the control system, the assistant system, and the flight strip system. The flight strip system, delineated by dashed lines, includes the assistant system and part of the control system, and provides an example of a heterarchical system.

The assistant system has a computer, a human assistant, and flight strip components. The computer receives its input from some other, remote computer and prints a flight strip, which the assistant then removes and places in an appropriate empty slot in the flight strip rack. The new flight strip is "cocked" (it sticks out a bit from the other ones), and there is often some communication, verbal or nonverbal, between the ATCO and the assistant. The assistant system can be considered as a work system, and its application domain is the flight strip application domain, which is that part of the control system (including the ATCO and flight strip rack) that the assistant system communicates with so as to perform work in this application domain. The work goals of the assistant system are to correctly place flight strips and to ensure the ATCO is aware of new strips in the flight strip rack. The critical point is that the work of the assistant system is not directly related to real aircraft or, in this case, to the flight chief, who is not responsible for checking the arrival of new flight strips.

It is usually an error, and probably a common one, for analysts to jump from identifying a failure to achieve the task goals with respect to one application domain and work system to concluding that a similar failure has also occurred with respect to a different, often more general, application domain. In other words, analysts unjustifiably assume that if a failure occurs in one part of a system, it will cause a problem in some other part of the system, usually a problem in a real world application domain, such as that part of the ATC system that includes aircraft flying around the sky (the application domain in Fig. 1.4). Only in a zero redundancy system is this assumption guaranteed to be correct. Zero redundancy virtually never occurs in real systems involving people because such systems are inherently noisy (i.e., human performance is variable and prone to error).

To avoid problematic assumptions, analysts need to study the relevant work system with respect to the goals they wish to refer to. Incorrect information on a flight strip, for example, may have no effect on the movement of aircraft if such errors are trapped and rectified before they affect the ATC work system goal (i.e., safe and expeditious movement of aircraft). Such errors may well cause concern to those involved in ATC system design. If only the communicative relationships between ATCOs and flight strips have been analysed, then rational efforts at solving work problems, (i.e., based on analysis), can only be directed at these relationships (e.g., attempts to redesign the ATCO–flight strip interface to reduce the probability of input errors). If the more general ATC system has been analyzed, then other design options rationally become available, such as improving the ATC work system's ability to detect and correct errors or reducing the number of possible transmission and transcription errors.

1.4 HUMAN-COMPUTER INTERACTION AND SOFTWARE ENGINEERING

The general assumption throughout this handbook is that the purpose of performing a task analysis is to improve computer systems and make them better tools for people's benefit. HCI's genesis as a distinct discipline arose in the early 1980s, and from this time HCI has been recognized as being inherently interdisciplinary and as requiring consideration of both psychological and computing issues as well as issues in many other areas (Diaper, 1989d, 2002a, 2002b).

HCI is a specialized subdiscipline of ergonomics. It is so for two related reasons. First, HCI restricts itself to the study of systems that have a computer component. Second, HCI's emphasis has been on human psychology far more than in traditional ergonomics, which started with an emphasis on matching human and device physical capabilities (see also chap. 28). This shift of emphasis is related to the view that computers are tools to augment human mental abilities whereas nearly all earlier tools augmented human physical abilities. In its early days, HCI was primarily concerned with human cognition, to the extent that Norman's (1986) classic chapter is titled "Cognitive Engineering," although wider issues related to organizational psychology were pioneered, for example, by Mumford (1983), and issues of social psychology have been promoted since the development of the field of computer supported cooperative work (CSCW) in the early 1990s. Furthermore, as chapter 29 argues, consideration of cognitive psychological issues is not sufficient, even within the psychology of individuals, and emotion (affect), life style, fashion, and so on, as important aspects governing human perception and behavior, also need to be taken into account (see also section 1.2.1).

HCI is an engineering discipline rather than a science because its goals are inherently practical and involve satisfying design criteria (Diaper, 1989d, 2002a). HCI design criteria, however, although they often involve aspects of computer systems, may also involve altering human cognition (e.g., by training people) and affect (e.g., by increasing their pleasure and hence motivation) as well as revising organizational processes and other aspects of application domains or work systems. As an example of the latter, improved ATC work systems have changed the ATC application domain so that more aircraft can be safely accommodated within the same airspace.

Although the roots of computer science are in mathematics (Diaper, 2002a), and there is scope for formal methods within HCI (Harrison & Thimbleby, 1990) and task analysis (e.g., chap. 11 and most of the chapters in part IV), HCI is most closely related to the computing field of software engineering. Diaper et al. (1998; Diaper, 2002a) go so far as to suggest that no distinction should ever have been made between software engineering and HCI because both are engineering disciplines concerned with the same types of systems and their difference is merely one of emphasis, with software engineering focusing more on software and HCI more on people.

1.4.1 Human-Computer Interaction

The discipline of HCI continues to be defined in two different ways, broadly and narrowly (Diaper, 1989d, 2002a). The broad view of HCI is that it is concerned with everything to do with people and computers. The narrow view is that it is concerned with usability, learnability, intuitiveness, and so forth, and is focused on the user-computer interface. These two views of HCI fit neatly with quite different definitions of work systems and application domains (section 1.3). In the narrow view, the application domain is the computer system, and the work system is the end user and other things in the user's environment. The users' tasks, in the narrow view, are to satisfactorily change the state of the computer system. The consequences of the computer's state change for other parts of the whole system are outside this narrow view of HCI. Hammer (1984) refers to interface design as a second-order issue and claims it is only relevant when two computer systems possess equivalent functionality (section 1.4.2.2; but see also chap. 30). Hammer's definition of functionality is firmly grounded in the real-world consequences of the automated office systems he discusses. User interface design issues are a major part of HCI, but a broader definition of the application domain and work system is needed if real-world consequences are to be taken into account.

There is a moral imperative associated with the broad view of HCI (Pullinger, 1989). It is certainly necessary for some discipline to consider the myriad effects that computers have on people, individually and collectively, and the supporters of the broad view claim that this discipline is and should continue to be HCI. The broad view treats user interface design as an important but subsidiary part of the whole HCI enterprise. Task analysis is at the core of HCI because of its emphasis on changes made to application domains, which are often in the real world (i.e., outside of the work systems). Such changes inherently concern system performance, which is what distinguishes task analysis from other more static, atemporal systems models (section 1.2.2.2).

1.4.1.1 The Psychology of Users

Notwithstanding the need in HCI to consider affective, social, organizational, and other such issues, most of the psychology in HCI and in current approaches to task analysis focuses on human cognition, and it is human cognition that is the main ingredient of user models in HCI. The point to recognize is that the cognitive psychology of people is much more complicated than, for example, the information-processing abilities of computer systems and that this creates a fundamental problem for task analysis. If an analyst cannot understand the operation of a basic system component (such as the human element), then it is nigh impossible to predict how the various things in a system will interact and produce the behavior of the system. A cognitive user model of some type is essential, however, as the Skinnerian-based behaviorist approach (chap. 3), which treats the mind as a black box and merely attempts to relate its inputs and outputs, has since the early 1960s generally been recognized as inadequate for either the explanation or prediction of human behavior. Fortunately (and provided that HCI is treated as engineering rather than science), it is possible to make use of psychological models of users that are good enough for engineering purposes, even if they are inadequate scientifically.

A vast range of psychological user models (see chap. 28 for a review) are employed across the range of task analysis methods available (for reviews of task analysis methods, see chap. 6, 22, 24, and 25). Admittedly, at one extreme the user models are implicit and resemble behaviorist models in their concentration on human behavior in a task context rather than on how mental operations cause such behaviors. The cost of ignoring mental operations is that, as soon as analysts move from describing behavior within an existing system to predicting behavior within new systems, they must rely on their craft expertise in human psychology. In one sense, all people are expert psychologists because they inhabit a rich, complex social world in which they must be able to understand and predict both their own behavior and that of other people. On the other hand, people do not do this very well, for otherwise a science of psychology would be unnecessary or at least could be simpler and more understandable.

Scientific cognitive psychology is not an easy subject to comprehend, even though it has concentrated on describing the mental architecture common to everyone (usually modeling the mind as an information-processing device) rather than on the contents of people's minds. As an example, it focuses on how memory works (architecture) rather than what is remembered (content). Chapter 15 provides an example of a sophisticated information-processing model of human cognition that can be used in task analysis. Although this interacting cognitive subsystems (ICS) model for cognitive task analysis (CTA) well illustrates the complexity and hence analytical power of scientifically based cognitive models and has demonstrated reasonable predictive adequacy over a quarter of a century of development, it is by no means universally accepted by professional cognitive psychologists, and there are alternative models, such as ACT-R (Anderson & Lebiere, 1998), that make similar claims to analytical and predictive capability. Furthermore, radically different cognitive architectures, such as those based on parallel distributed information processing (Hinton & Anderson, 1981), also have their supporters among professional cognitive psychologists.

A useful contrast can be made between the ICS approach and that described in chapter 16, where it is assumed, probably incorrectly (Diaper, 1982, 1989b), that people have access to their own mental processes and can describe these when interviewed appropriately. As chapter 19 points out, there is often a considerable difference between what people say and what they do (see also Diaper, 1989a, and chap. 28). Although unsatisfactory from a scientific perspective, the assumption made in chapter 16 may well be adequate for some HCI engineering purposes, as people's beliefs about their own minds are likely to affect their behavior even if such beliefs can be experimentally demonstrated to be inadequate or even incorrect. Chapter 16 reports a considerable degree of accuracy in the recall of relevant events in some of the interviews collected on critical incidents.

Between the extremes represented by implicit psychological models and models like ICS are the more popular cognitive models associated with such task analysis methods as HTA (chap. 3) and GOMS (chap. 4). Note that HTA is based on the description of tasks as a hierarchy of goals and that, as Annett says in chapter 3, it "differs radically from earlier methods of task analysis by beginning, not with a list of activities, but by identifying the goals of a task." It is unfortunate that some users of HTA continue to confound psychological goal states with descriptions of behavior. Consider Annett's first example: "the goal hierarchy for an acid distillation plant operator's task" (Fig. 3.1) This hierarchy has a high-level goal. ("Inspect instruments & locate fault") and lower level ones (e.g., "1.2.4 Start standby pump"). Given that a videotape could be made of the operator starting the standby pump it is easy to understand why some people forget that in HTA the object is to describe psychological goals and not behavior. Remember Annett's warning, already quoted in section 1.3.2: "Simply to list actions without understanding what they are for can be misleading" What is really needed in HTA and in task analysis generally is a language that differentiates psychological things from observable behavior so as to avoid this potential source of confusion. Note that one alternative would be to assent to philosophical behaviorism (Atkinson, 1988), which should not be confused with the Skinnerian sort. Philosophical behaviorism denies that correctly specifying psychological things is possible, and it thus can allow observable behavior to stand for unknown psychological states (Diaper & Addison, 1991). As Diaper (2001b) pointed out, philosophical behaviorism has not become popular. Nonetheless, it is discussed further, from a more radical perspective, in chapter 30.

In chapter 4, Kieras states that GOMs is intended as a method to be used after a basic task analysis has been carried out and that its purpose is "to provide a fomalized representation that can be used to predict task performance well enough that a GOMS model can be used as a substitute for much (but not all) of the empirical user testing needed to arrive at a system design that is both functional and usable." GOMS employs hierarchical decomposition so as to identify, among other things, "primitive operators," which may be internal (cognitive) or external (observable behavior). It then assigns an estimated execution time to each primitive operator, usually 50 ms for internal ones and longer durations, based on task observation data, for external ones. GOMS uses a "production system model for human cognition." This model, although undoubtedly a misrepresentation of human psychology, predicts task performance time by simply summing the time associated with each primitive operator. That is, it treats task processes as independent of each other, although GOMS-CPM (Critical Path Method; see chap. 22) is claimed to incorporate some parallel cognitive processing. GOMS has enjoyed considerable success over the years at predicting both task performance time and errors. The GOMS psychological model of users is thus good enough, usually, for engineering purposes.

In summary, there is no escaping the need for task analysis to include a psychological model of the human parts of a system. The model used, however, will be adequate if it merely approximates the real, albeit still unknown, psychology of people. After all, HCI is an engineering discipline, and its practitioners must do their best to satisfy the practical criteria associated with particular projects. A number of chapters in this handbook (e.g., chaps. 3, 7, and 17) suggest that task analysis requires expert practitioners. One reason to believe that it does is that selecting and then using one or more psychological models requires experience and craft skills (Diaper, 1989e) and should not be confused with merely following the steps of a method, no matter how well specified or complicated the method is.

1.4.1.2 The Psychology of Computing Industry People

Software engineers are people too! So are systems analysts, designers, programmers, and even managers. A specialized branch of HCI was developed in the 1980s and became known as the psychology of programming (PoP), which is something of a misnomer these days, as the field covers all sorts of people involved in the computing industry. PoP has the potential to become a vast field of study, but it remains fragmented and underresearched (Diaper 2002a). An understanding of computing industry people (CIPs) is of importance to HCI and for task analysis for two reasons. First, as the producers of computer systems, CIPs have some influence over the computer systems that are delivered. Second, CIPs are users of methods and CASE tools and of the results of these.

Norman's (1986) classic HCI model identifies models of two types: users' models of the system image and designers' models of the users' models of the system image. In reverse order, the system image is everything that the end users can perceive about a computer system by interacting with it, including its manuals, any training provided, and so on. These users construct their users' models on the basis of the system image (see also chaps. 18, 19, and 24). The computer system's designers have their own models of the users' models of the system image, and Norman suggests that one major source of problems with delivered computer systems is that there has been a mismatch between the designers' models and the users' models.

Furthermore, some of CIPs' influence on the development of a computer system is implicit; that is, people make decisions and perform actions without understanding their basis or perhaps even recognizing that they've made a decision. Various different styles of working, whether in systems analysis, design, programming, or management, provide numerous examples of consequences of development that are style dependent and not understood by the workers. Carroll (2000) makes a case that design in particular is difficult, is not routine, and is creative, and just the last of these implies that no two designers or design teams will ever produce quite the same thing. Indeed, the dispute between Bench-Capon and McEnery (1989a, 1989b) and

Barlow et al. (1989) (see sec. 1.3.1.2) hinges on the latter's view that delivered computer systems have many properties that are implicitly caused by those who do the designing and building of them, including how these people are managed. The importance of this for HCI and task analysis is that it partially explains why the performance of computer systems is more complicated than ever described in their design and development documentation. This chapter has already suggested that people, as system components, are very complex (section 1.2.1), and perhaps the complexity of any reasonably sized computer system has also been generally underestimated.

The second reason for introducing CIPs into this chapter is that they are users of methods and CASE tools, including task analytic ones, and users of the results of the methods, perhaps because some methods, including task analytic ones, need to be carried out by those with the appropriate craft expertise (sections 1.4.1.1 and 1.6 and chaps. 3, 4, 7, and 17). Diaper (1989e, 2002a) suggested that there is a problem delivering HCI methods to the computing industry. One of the primary reasons TAKD failed (Diaper, 2001b) is that it was too complicated to do; certainly a CASE tool was essential (see also sections 1.4.2 and 1.5.1.5). Diaper has even suggested that, when designing a method, the designers should determine the method's design requirements by modeling its CIP users as much as by looking at the domain of the problem that the method addresses. TAKD, for example, was originally intended to identify the knowledge people required to perform IT tasks and so form the input to IT syllabus design. Unfortunately, it gave too little consideration to TAKD's other potential users. Undoubtedly this caused TAKD to be too complicated to be successfully delivered to the computing industry, notwithstanding its flexibility and analytical power. CASE tools may be a good way to facilitate the industrial uptake of methods, they may sometimes be necessary, perhaps for methods such as TOOD (chap. 25), but they are not sufficient. The delivery of task analysis to the computing industry is further discussed in chap. 30.

1.4.2 Task Analysis in the Software Life Cycle

The discipline of software engineering developed from the recognition in the late 1960s that there was a "software crisis," namely, that the demand for quality software had far outstripped the capabilities of those able to produce it. The software crisis remains with us today. Most software engineering solutions have been anthropocentric (Diaper, 2002a) and have attempted to increase software engineering productivity and improve the volume and quality of the software produced by providing methods of working, organizational as well as software-oriented ones, and building CASE tools to support some of these methods. Chapters 23 and 24 review a number of task analysis CASE tools, and most of the chapters in part IV of this handbook, along with others, such as chapter 7, describe CASE tools that support particular task analysis methods.

The most widely cited early attempt to produce a structured software development method is now usually referred to as the "classic waterfall model" (e.g., Pressman, 1994; Sommerville, 1989). This model, describes the major stages of software development using labels such as requirements, design, coding, testing, and delivery. The waterfall aspect is the strict sequencing of these main stages. Although it is always possible to return to an earlier stage to make changes, the development must again cascade through all the stages below that one. It is generally accepted, after much practical software engineering experience, that the earlier in the waterfall any changes are made, the greater is the additional cost to a project. Often the costs of change are suggested to be an order of magnitude greater for each stage further back in the sequence. The waterfall model is not now used in most software engineering projects, primarily because it fails to describe how software engineers actually work, but it continues to provide a convenient vocabulary to describe the software life cycle.

The waterfall model was followed by many structured software development methods, such as Jackson Systems Design (JSD; e.g., Cameron, 1983) and subsequently, the overly complicated Structured Systems Analysis and Design Method (SSADM; e.g., Ashworth & Goodland, 1990; Downs, Clare, & Coe, 1988; Eva, 1994; Hares, 1990). A method of this type defines one or more notations and provides relatively simple descriptions of how to use these. One general problem is that methods are difficult to describe. Methods involve processes but most natural languages are declarative, that is, they are much better at describing things than how things are transformed (Diaper, 1989b, 2001b; Diaper & Kadoda, 1999).

As an alternative to structured methods, various prototyping approaches have been developed. In a prototyping approach, an incomplete version of the software is developed and tested in a series of iterations. The hope, of course, is that a satisfactory system will eventually result (see section 1.2 and chap. 7).

A question that is always relevant is, 'Where will task analysis be used in the software life cycle?' The old HCI slogan, "Get the human factors into the earliest stages of a project," still applies (e.g., Lim & Long, 1994). In particular using task analysis early and repeatedly throughout a project can improve quality and reduce costs. It has also been argued (e.g., Diaper, 1989c) that the cost of using task analysis is greatly reduced when it is used throughout a project, as the initial effort of producing the systemic description of the world can be amortised over the life of the project. For example, if task analysis has been used at the requirements stage, it should be cheap to use the same model, with appropriate minor changes to reflect the development of the new computer system, for evaluation purposes, and quality should be improved because similar problems are being addressed at both these stages (see section 1.5.1.1).

1.4.2.1 Task Simulation and Fidelity

Howsoever one collects the data for a task analysis of a current system (sections 1.2, 1.5.1.4, and 1.5.2.1), all one ever has is a simulation of the real tasks of interest (Diaper, 2002b; chap. 30). There are three reasons for this. First, as Carroll (2000) has pointed out, generally there are potentially an infinite number of tasks carried out by different people that can be studied, but only a small number can be selected for analysis. Second, as mentioned in section 1.3.1, only a small part of a system can ever be observed so that the task data are always incomplete. Third, there is nearly always a Heisenberg effect: the act of collecting the data alters what is being studied. The only circumstances in which a Heisenberg effect would not occur is where the people or other things being observed are not affected (and, in the case of people, not even aware they are being observed). Thus, observation and other ways of collecting data about tasks are not easy and require considerable craft skill (section 1.5.1.4; Diaper, 1989a). In the case of a proposed future system (section 1.2), of course, the tasks can only be simulations, as broadly define in Life, et al. (1990).

It is therefore necessary to realize that task analysis always deals with simulations. *Fidelity* (a closely related term is *validity*) refers to the degree to which a simulation models the assumed real world (Section 1.2). (From a solipsist perspective, it is the degree of agreement between two models of the assumed real world.) Diaper (2002b) argued that fidelity varies widely across task analyses. Fidelity can be reasonably high when an existing system or prototype and its real end users in their natural environment are studied. Fidelity can be very low when use scenarios (short prose descriptions of the performance of a putative future system) are used (Carroll, 2000; chap. 5), which is not to say that these are not helpful, particularly in the earlier stages of the software life cycle. Between these extremes are approaches that are intermediate in fidelity and that are usually recognized as and called simulations. Many of these are paper based or use only a user interface without the back-end application software (e.g., Diaper's, 1990a, Adventure Game Interface Simulation Technique [AGIST]). The Wizard of Oz simulation technique provides higher fidelity simulations and has been widely used, for example, for studying natural language–processing computer systems that cannot yet be built (e.g., Diaper, 1986; Diaper & Shelton, 1987, 1989). In a Wizard of Oz simulation, the natural language–processing capabilities of a future computer system are, unknown to the users tested, simulated by a person in another location, and with sufficient care in design, such a simulation can be very convincing to users. Undoubtedly there is a strong correlation between fidelity and the point at which task analysis is used in the software life cycle, with fidelity usually lower the earlier task analysis is used (see, e.g., Benyon & Macaulay, 2002).

1.4.2.2 Functionality

Within the computing disciplines, the term *function* and its derivatives, such as functional and functionality, are polysemous (they have more than one meaning). Functionality, for instance, refers to features available on the user interface and to capabilities of the back-end software that support the achievement of goals. These two types of functionality share an asymmetric relationship. The first is really a user interface design (UID) issue, supported by the narrow view of HCI (section 1.4.1), and it is important because a computer system's back-end functionality is useless if people cannot access and use it easily. On the other hand, the back-end functionality has to be there first. Questions about what a computer system can do as part of a work system are crucial to HCI and are more important than interface issues (section 1.4.1). Although Sutcliffe (1988) is right that good UID can save a functionally sound computer system, he is wrong to suggest that better user interfaces can save poor software. In HCI, functionality is a property of any thing in a work system, but it is usually ascribed to a work system's agents (section 1.2.1; see also chap. 30).

Functionality is typically thought of as static, as a property of a work system's agents and not of its performance. Although this may be a helpful way to think about functionality, it is obviously wrong, as there will often be functionality performance issues. For example, there are many things that cannot be computed simply because the program would take too long to run; encryption algorithms rely on this. At the other end of the spectrum, the functionality of a real-time control system is severely constrained by the sequence and timing of events as well as complexities in the application domain beyond the system's direct control. Given that task analysis is primarily concerned with the performance of work, one of its potential contributions is to get analysts to realize that functionality applies to performance as well.

One specialized use of task analysis is for functional allocation (see chaps. 6, 22, and 24), which involves dividing the labor among the agents and determining which other things are used and how. Automating everything possible and leaving the remainder to people was soon recognized by ergonomists as a recipe for disaster.

As an example of how functionality can be allocated differently, consider where the intelligence to perform tasks is located in work systems that include expert systems and work systems that include hypertext ones (e.g., Beer & Diaper, 1991). Both types of computer system can be formally described by graph theory as having an architecture of nodes connected by links that can be traversed. In an expert system, the intelligence involved in link-traversal tasks resides in the expert system's inference engine, and the users are reduced to being suppliers of data to the expert system. In a hypertext system, however, the intelligence required to traverse links is the users', and the computer part of the system is passive (section 1.2.1). Web search engines provide one means of allocating some of the link-traversal task functionality to the computer system.

Functionality is obviously such an important concept in software engineering, HCI, and task analysis that it is odd that it has not be more fully discussed within the research literature. Although the term functionality is not absent from this handbook's index, for example, it is not one of the most widely used terms herein. Chapter 30 returns to the topic of functionality and takes a more radical stance.

1.4.2.3 Implementation Independence, Abstraction, and Goals

David Benyon and the author of this chapter have long been engaged in a friendly public discussion about whether and in what way task analytic descriptions are independent of any particular design or implementation (Benyon, 1992a, 1992b; Benyon & Macaulay, 2002; Diaper, 2002c; Diaper & Addison, 1992). Similarly, several chapters of this handbook disagree as to whether task analysis should represent realistic, concrete examples of tasks (e.g., chaps. 2 and 3) or generalized, abstract, higher level task descriptions (e.g., chaps. 4 and 7). Most task analysis methods are capable of doing both, of course, but there appears to be little agreement as to how close a task analysis should be to a specific implementation in a particular environment, and so on. Obviously, when task analysis is used prior to design, there must at least at some low level of abstraction and some independence between the task analytic representation and the design to be subsequently produced. When there is an existing system or prototype on which to base a task analysis, however, the question arises as to how closely the task analysis should be tied to use of the system (i.e., what degree of the fidelity the task simulation should possess; section 1.4.2.1). This issue has almost certainly caused confusion in the task analysis literature. Part of the confusion can be resolved by considering where in the software life cycle task analysis is to be used (sections 1.4.2 and 1.5.1) and what its purpose is (section 1.5.1.2). When used to evaluate a current system, a task analysis might be very concrete, describing specific things and behaviors in a detailed way. In contrast, in a creative design situation (e.g., the family-wear scenarios in chap. 5), the task analyses must be at a higher level of abstraction because the design has not yet been fleshed out. Furthermore, some writers, such as Benyon and Macaulay (2002), have argued that further abstraction is desirable. Diaper (2002b), however, suggested that scenarios such as those of Carroll are not proof against premature commitment to some detailed design. In contrast, for example, although Task Analysis for Error Identification (TAFEI; chap. 18) is able to evaluate the design of devices prior to their implementation, the method is highly device dependent.

The concept of goals, at least people's goals, occurs in every chapter of this handbook, although only this chapter (section 1.3.2) and chapter 26 state that things other than people can have goals. Although there is a strong case for considering goals to be essential for explaining the observable behavior of system components (sections 1.3.2, 1.4.1.1, and 1.5.1.5), there appears to be a second, generally implicit argument that describing goals is a means of abstracting away from an implementation-dependent description. The argument here is that describing what people want to do (their goals) is easily separable from what they have to do (the tasks they must perform) to achieve these goals. Section 1.4.1.1 suggested that in HTA (and probably many other task analysis methods) one source of confusion is that descriptions of psychological states such as goals are not sufficiently different from descriptions of observable behavior. A related concern is that if task analysts believe that by describing goals they are safe from making premature design commitments, they might make such commitments unwittingly. Furthermore, if people naturally think in relatively concrete scenarios, as proposed by Diaper (2002c), then unwitting premature design commitments are quite likely. Overall, the lower the level of abstraction (i.e., the more detailed the task analysis), the greater the chance of such problems occurring. The issue of the abstraction role of goals is dealt with in chapter 30.

1.5 DOING A TASK ANALYSIS

There are many types of projects that involve the design, development, delivery, and maintenance of computer systems. All these projects have a human element in that the projects themselves are staffed by people (section 1.4.1.2), but all also have a broader HCI element, as delivered computer systems always have an impact on some people's lives. Furthermore, given that task analysis, because of its concern with system performance issues, is at the core of HCI (sections 1.1 and 1.4.1), virtually all computer-related projects will involve some form of task analysis. This conclusion is not invalidated just because the majority of computing projects do not explicitly recognize the premises of the argument: Any computing project involves people inside and outside the project; such people are a main focus of interest for HCI; HCI is concerned with performance and hence task analysis. It would undoubtedly facilitate clarity of thought, however, if the premises and conclusion were accepted in every computing project so that the various roles of task analysis could be identified in each project.

One reason, of many, for identifying the roles of task analysis in computing projects is that task analysis cannot be fully automated. Analysts must make many decisions that are subjective (neither right or wrong) but may be more or less useful. Task analysis CASE tools (sections 1.4.1.2 and 1.4.2 and chaps. 23 and 24) can make analysts' decision making easier in many ways, but some "judgment calls" are unavoidable. Perhaps a long way in the future, approaches such as Subgoal Templates (chap. 17) might lead to some artificial intelligence–based subjective decision making, but this is far beyond current AI capabilities and may never be deemed entirely desirable. Thus, given that computing projects will involve task analysis and that task analysis will be iffy and introduce indeterminacy (aka noise) in project management, then identifying the use of task analysis in projects must be a valuable aid to project management and project cost estimation. All this boils down to the proposition that computing projects will, nearly always, involve task analysis, so they might as well recognize task analysis, and if they are going to recognize it, then they might as well call it task analysis as well.

This handbook, though by no means exhaustive, contains a considerable number of different task analysis methods suitable for different purposes at different stages in projects. What most distinguishes these methods are their later analysis stages (see section 1.5.2).

1.5.1 What For and How?

Whether well specified or not, all software engineering projects involve some form of development method. Obviously, the first question is, 'Where could task analysis be used in the project?' but answering this may be far from simple. Many activities in software engineering methods are not task analytic, so perhaps the first real question for each step in a method (section 1.4.2) is to ask, What can task analysis directly contribute? Sometimes the answer will be "nothing," as is usually the case when encoding a design and for some forms of code testing. This answer is reasonable, because task analysis only has an indirect contribution to such project stages via other stages, such as requirements and design specification and functional and user testing.

Having identified where task analysis can contribute to a project, analysts commonly leap to the question, What task analysis method? This is an error, and it can make the results of the task analysis unusable. The author has been as guilty of this mistake and in his case it resulted in many years of research trying, for example, to bridge the gulf between requirements and design (e.g., Diaper, 1990a, 2001b), because the requirements representations from the task analyses were incompatible with design ones. The solution, once this problem is recognized, is to work backwards at the project level and iterate around at the individual stage levels.

1.5.1.1 Working Backwards and Around

Computing projects are going to produce something because they are engineering projects, although the something produced may not be a computer system or software but instead a training program, manual, or management system. Indeed, at the start of some projects, even the type of solution may not be known, such as when using SSM to address soft problems rather than hard ones. Once the stages of a project have been identified so that the "Where could task analysis be used in the project?" question can be addressed, it is best to start with the relevant final stage of the project and work back through the stages asking this question. This strategy is the opposite of current practice, but it is sensible because in most projects the relevant, nearly final stage, will involve some form of testing of the performance of the project's product. Performance, of course, is central to task analysis, so this stage will almost certainly have a yes answer to the question. Furthermore, in an ideal development method, everything specified in the earlier stages of a computing project ought to be evaluated before the product is released, which is a consequence of the principle that everything specified at one project stage should be covered in subsequent project stages. The same principles apply to prototyping cycles as well.

Having identified the stages to which task analysts can contribute, analysts can address a whole host of other questions at each stage. Ultimately, the project stage that will require the most extensive, detailed, and expensive task analysis should determine how task analysis will be used in other stages so that the costs can be amortized over these stages (section 1.4.2). Although a common assumption is that the stage needing the most extensive task analysis is a requirements one, it might just as plausibly be a near final performance evaluation one, and it could be even some other stage, depending on the type of project.

Starting with the stage that will require the most extensive task analysis, the critical questions are as follows:

1. What do you want to know from the task analysis?
2. What is the most appropriate task analysis output format?
3. What data can be collected?
4. What task analysis method?

Although it is tempting (and common practice) to ask these question as they arise in the carrying out of the work, it is better to ask them initially in the order given above. The questions are not independent, so it is essential to iterate around them to refine the answers to each. Just as one should never conduct a scientific experiment without knowing how the data will be analyzed, one should never conduct a task analysis without addressing these questions. In the worst case cost scenario, a great quantity of task analysis data will be collected and analysed but never used. This is costly for two related reasons. First, collecting data is itself expensive. Second, large volumes of data not only take a long time to analyze but generally generate confusion among the analysts. It is easy to suffer two combinatorial explosions, of data collection and of analysis, sometimes to the extent that what is produced is less useful than what would have been produced if a more modest but better planned approach had been adopted.

Once the above questions have been addressed for the project stage that will make the most extensive use of task analysis, the same questions can be addressed for the other stages. Generally, the answer to question 4 should be the same for each project stage, and likewise for question 3, for the same data collection method should be used at each stage even if the emphasis on the data and the degree of detail vary. Questions 1 and 2 are stage specific. The rationale for using a consistent task analysis approach is that it will (1) amortize the costs over the project, (2) maintain or improve the relationship between project stage, and (3) transfer expertise across the stages.

Planning across the whole project is the key, but such planning needs to be approached iteratively, and the analysts need to be prepared to return to each project stage and modify their planning. No doubt sometimes a single task analysis approach cannot be adopted across an entire project, but use of a single approach should be a goal that is abandoned reluctantly. In many real computing projects, different personnel work on different project stages, and insufficient communication and management across the stages is a common problem. In a typical case, the design and implementation team hand over their prototype to the human factors team for usability testing, but the latter use an explicit task analysis method that radically differs from the, perhaps implicit, method used for requirements and design specification, making the results of the usability testing partially if not totally inapplicable to the design. One of the strengths of Coronado and Casey's use of task analysis (chap. 8) is that a similar task analysis is used iteratively throughout the stages of a project.

The following four sections address in more detail the four key questions discussed above.

1.5.1.2 What Do You Want to Know From the Task Analysis?

For each stage in any well planned computing project, this must be the critical question. Given the vast range of projects and the variety of development methods, offering general advice is nigh impossible. Worse, this is a very difficult question to answer, and the suspicion must exist that in real projects it is often dodged on the basis that the task analysis is exploratory and will highlight interesting issues. Unfortunately, in general task analyses won't have this result unless the analyst looks for "interesting issues" in the analysis. Admittedly, some issues may arise serendipitously, as the act of performing any sort of analysis often enforces a discipline on the analyst that leads to insights that would not otherwise occur, as also argued in chapter 4. On the other hand, basing an engineering approach on hoped-for serendipity is not a trustworthy strategy. The trick is to be resigned to repeated refinement of the answers to this question as the others are considered. In other words, start with this question but don't expect to get it right the first time.

1.5.1.3 What Is the Most Appropriate Task Analysis Output Format?

One of the strengths of structured approaches to development in software engineering is that different stages use one or more different representations. For example, among other representations, DFDs, Entity Life Histories (ELHs), Entity Relation Diagrams (ERDs), and Process Outlines (POs) are used in SSADM (section 1.4.2), and use cases, activity diagrams and interaction diagrams, are used in UML (see chaps. 7, 11, 12, 19, 22, 23, 24, and 27). It would therefore seem desirable for task analysis output to be in the form of one or more of such software engineering representations or be easily related to them. This desideratum is not met by many task analysis methods, although, in this handbook, methods developed for industrial application (part II) and those discussed in part IV have been designed or adapted to relate to software engineering representations. One potential disadvantage of task analysis methods that easily relate to software engineering representations is that they often require a specific software engineering method to be adopted or demand changes, often considerable, in the chosen software engineering method. In either case, the needed revisions may be unacceptable or costly for companies that have invested heavily in a particular software engineering approach not explicitly supported by task analysis, although such companies might benefit in the long term by switching to methods that do have an explicit task analytic component.

The core of the problem is the historical separation of software engineering and HCI (section 1.4). Many task analysis methods were developed by researchers with a psychological background, and these methods and their outputs often do not integrate well with those of software engineering. Furthermore, adapting task analysis methods post hoc so that their output is in a software engineering representation seems to be difficult. Diaper (1997) reported that an attempt was made to modify TAKD so that its output would be in the form of ELHs, but this attempt was subsequently recognized as a failure (Diaper, 2001b). On the other hand, the Pentanalysis technique has been successfully used to directly relate task analysis to DFDs (Diaper et al., 1998).

Although asking how task analysis output contributes to a computing project is essential, there are many cases where, even though the output does not easily fit with the software engineering approach, undertaking one or more task analyses is still valuable. One reason is that software engineers and systems analysts, for example, often do task analysis implicitly and poorly (section 1.2.2.2; Diaper et al., 1998), and doing a proper task analysis explicitly is nearly always bound to lead to an improvement in quality (at a cost, of course). Another reason is that the act of doing a task analysis can have benefits not directly related to the collected data. First, preparing for a task analysis and building models of the world can improve the similar models that are essential in software engineering. Second, collecting task analysis data forces analysts to look at their world of interest with care and often in more detail than is common in software engineering projects. Third, the analysis of the task analytic data usually causes further reappraisal of the data or the collection of new data. The author of this chapter has frequently commented (e.g., Diaper, 2001b) that in more than 20 years of doing task analyses he has never got the task analysis data representation of activity lists (section 1.5.2.2) correct the first time, and TAKD's method explicitly recognized this in its analysis cycles. In summary, no one wants to expend effort, time, and money conducting task analyses that cannot significantly contribute to a project, but the contribution of a task analysis may be considerable even if it is indirect and the analysis output does not directly map to the software engineering representations that the project uses.

1.5.1.4 What Data Can Be Collected?

One of the myths about task analysis is that it involves the detailed observation of tasks. Although in some cases this may be so, task analysis should be able to use and integrate many types of data (Diaper, 1989c, 2001b; Johnson, Diaper & Long, 1984; chap. 3). Integrating different types of data is facilitated by treating data from a performance-based perspective (section 1.3). Of course, disentangling the identification of what data can be collected from the methods of collecting it is not easy. The effort is worthwhile, however, so that the analysts are at least aware of what sorts of data might have been missed.

Apart from the observation of performance, task analysis data can be gathered from interviews and questionnaires, from all sorts of documentation, from training programs, and from consideration of existing systems. Part of the skill of the task analyst is in recognizing how data conflict and being able to see, for example, that the official account of how something works (e.g., according to training systems or senior management) is at odds with actual practice. Resolving such conflicts requires social skills and often raises ethical issues. Task analysis, because its primary focus is performance, tends to be socialist. After all, it is the workers, those who actually perform the tasks of interest, who are the task experts and who will usually be those most affected by changing the current system. Indeed, many IT system developments have failed because the workers were not properly carried along. Bell and Hardiman (1989) reported on one such case:

> We even have one example of a highly successful knowledge-based system which is saving its owners many millions of dollars a year. The designers consulted some of the potential users of the system when the system was being built, but as the designers only worked 9 am to 5 pm they only consulted the first- and second-shift users. When the system went live, the night-shift workers refused to have anything to do with it: the system was only being used by two-thirds of the work-force. (p. 53)

Where there is conflict between the views of managers and workers, negotiating the difference is essential, as a new or revised system will likely fail if it is based on an incorrect model of the world (note that the official model will often be less correct). Managers do have a right and duty to manage, of course, but some react defensively to any suggestions that their view of the world is not the only possible one. The author's experience working as a consultant to industry is that the workers who perform the tasks of interest are often enthusiastic, sometimes overly so, about providing data for task analyses because they perceive it as a way to advantageously influence their future work. It is also worth noting, in passing, that people are usually quite enthusiastic about talking about their job anyway, because they spend hours a day at it and it tends to be a taboo subject outside of the work environment and often with everyone except their immediate coworkers.

Although there are always general issues concerning the fidelity of data (section 1.4.2.1), one particularly important aspect of data fidelity is the data's level of detail. Obviously, collecting highly detailed data that are not used is hideously expensive, but having to return to data sources to collect more data can also be very expensive (Diaper, 1989a). There are two sets of related problems having to do with the fidelity of data. One arises for analysts who use a deductive (top-down) task analysis decomposition method such as HTA, and the other confronts those who use inductive (bottom-up) methods such as were characteristic of TAKD.

First, because HTA is a deductive analysis method, even though it has its P × C stopping rule (chap. 3), the analysts do not know the required level of data detail until they have done the HTA analysis. Unless the analysts are highly inefficient and collect quantities of data that remain unanalysed, they will have to switch back and forth between data collection and analysis. The problems of such iteration are relatively obvious when the data are derived from interviews or questionnaires. In such cases, if a topic is not raised, then the data required will be absent, and the analysts will have to return to their data sources. There are expenses, however, even with data sources that are relatively complete, such as video recordings of tasks or extensive documentation (e.g., training manuals), as these sources have to be returned to and reanalysed. To take the case of video recordings, where missed data are available on the recordings (and sometimes the data are simply not captured), the recordings have to be watched again. Making video recordings of tasks is easy, but it is time consuming and tedious to extract data from them, say, about 10 hours of analysis for every hour of recording. Note that a video recording is not itself an observation but merely a medium that allows analysts to observe events outside of their real-time occurrence (Diaper, 1989a).

A related problem that plagues inductive methods is that the level of data detail must be selected prior to the data's analysis. The only safe and efficient approach here is to try to ensure that the initial data collected are as detailed as is likely to be necessary. Although collecting detailed data is expensive, the advantage of working from details upward is that only the pertinent data need be analyzed. Inductive methods are therefore particularly appropriate for analyzing data from media such as video recording.

Deductive and inductive analysis methods also face task sampling problems (chap. 30). One substantial problem for inductive methods is that, until the analysis has been done, empirically based principles to guide sampling are lacking. There are two types of sampling problem given a range of tasks have been sampled, (1) tasks not sampled from within the range, and (2) possibly relevant tasks outside the sampled range. Although neither problem is amenable to a guaranteed solution, solving the latter is particularly difficult and must rely on the cunning and imagination of the analyst; the former sometimes can be dealt with using some form of statistical analysis. Chap. 8 describes commercial situations where analysts' access to people and their tasks are very limited, and chapter 2 prioritizes the desirability of sources of task analysis data, recognizing that it is not always possible to adequately sample tasks with the ideal people and in the ideal environments.

With deductive methods, it is theoretically possible to establish what data to sample as the analysis progresses to lower levels of detail. What Green (1990) calls "viscosity" however, comes into play: as the analysis progresses, it becomes increasingly hard and expensive for the analysts to change earlier parts of the analysis. Viscosity is less of a problem with induction, and is particularly well illustrated by machine induction, as it is widely recognized that most things that can be validly induced from a data set are not what is wanted. The equivalent problem with deduction is not widely recognized, however, perhaps because deduction engines such as program compilers are easy to produce. Of course, they are easy to produce because we don't care exactly what is deduced but only that it is a valid version. That is, apart from the issue of functionality (e.g., that some compilers, are optimized for execution efficiency, others for debugging), we don't care that different compilers produce different machine codes. In contrast, in task analysis we do care what is deduced at each level of an analysis, and thus similar sampling and analysis problems occur with both inductive and deductive approaches.

Although in a minority, there are application areas where a complete set of correctly performed tasks can be analyzed, but usually not a complete set of incorrectly performed ones. There are two such application areas: (a) where the primary tool to be used is of such limited functionality that it can only be used in one way to perform a task (e.g., Baber and Stanton's digital watch, discussed in chap. 18); and (b) in some safety critical systems, defining safety very broadly (Diaper, 1989d) where correct task performance is enforced. Note that enforcement is only likely to be close to successful when functionality is relatively limited, that is, not in complex, flexible safety critical environments such as ATC (section 1.3.2 and chap. 13).

Carroll (2000; see also Diaper, 2002b, 20002c) is correct however, that there are potentially an infinite number of ways that different people may perform a range of tasks more or less successfully and efficiently, and thus it is necessary to sample tasks for analysis. There are two basic approaches to task sampling: using opportunity samples and doing selective sampling. Both may be used in a project, but they do have different consequences for analysis, as noted below.

In an opportunity sample, which is most frequently used with some form of relatively high fidelity data recording, such as video recording, the analyst, having carefully arranged everything beforehand, of course, simply arrives and records whatever tasks are going on in the workplace. This is a particularly good approach when the tasks or subtasks of interest are of a relatively short duration and repetitive. ATC might be a good example, although even when we used opportunity sampling a decade ago in the Manchester ATC tower, we did try to ensure that, in the 1 hour of live video recording that we took, the workload did vary, as we were aware that ATCOs and flight chiefs work slightly differently depending on their current workload. For example, when they are busy, they are more likely to give instructions to an aircraft to fly to a radio beacon rather on a specific geographical heading. The great advantage of opportunity sampling is that it allows an analysis of the frequency of tasks and task components. The disadvantage, of course, is that some critical but rare tasks or subtasks may not be observed, and information on these must always be sought from the task experts.

Although selective sampling of tasks has the disadvantage of making frequency data unavailable, it does result in a principled coverage of the range of tasks sampled. The sampling can be any combination of frequent tasks, complex tasks, important tasks, and error-prone tasks. Obviously the task analyst needs to understand the possible task range before collecting the data, and the most common source of such information consists of those who currently perform the tasks or might do so in the future. Usually one gets this sort of information by interviewing people, but care needs to be taken with such interviews because people do not usually classify the tasks they do in the way the analyst is likely to. Critical incidence reports (chap. 16) constitute one rich source of complex, important, and error-prone tasks, although there are potential problems with how well people recall critical events (section 1.4.1.1 and chap. 28). Another useful source of task samples consists of training schemes, which generally cover frequent and important tasks well but are often less helpful for identifying complex tasks (because subtasks interact) and error-prone task and error recovery tasks (which are sometimes not recognized at all in training schemes).

The final warning on task data collection is that, whatever method is used (e.g., generating scenarios, ethnography, interviews, and task observation), adequate collection of data depends on the craft expertise of the task analyst. One weakness in the contents of this handbook is that, unlike its predecessor (Diaper, 1989a), it does not contain a chapter that explicitly addresses task data collection methods, although many chapters provide advice and examples relevant to the analysis methods they describe. There are lots of tricks of the trade for any of these data collection methods, but the key to success is always thorough preparation in advance of data collection.

1.5.1.5 What Task Analysis Method?

As noted in section 1.5.1.1, the question of what task analysis method to use should be the last one addressed and not the first, but the questions covered in the subsections above should be iterated, as they are not independent. A task analysis method in this context is a method that converts task-orientated data into some output representation for use in a project. A major goal of chapter 22 is to help analysts select task analysis methods appropriate for their purposes.

A not unreasonable suspicion is that, in practice, people either choose a task analysis method with which they are familiar or they use something that looks like HTA. As chapter 3 makes clear, there is more to HTA than merely the hierarchical decomposition of task behaviors, and there it is argued goals rather than behaviors are what are decomposed in HTA (see also sections 1.3.2 and 1.4.1.1). Indeed, this handbook's editors rejected more than one chapter because it simply glossed over the relevant analysis method, often referring to it merely as a "basic task analysis," although this concept does still appear in a few chapters that address other aspects of task analysis in detail (e.g., chaps. 4 and 7). Section 1.5.2 does attempt to describe what may be thought of as a "basic" task analysis in that it tries to describe the initial steps involved in virtually any task analysis before a specific analysis method is applied to the data.

Although analysts understandably would prefer to use a task analysis method with which they are familiar, giving in to this preference is generally a mistake and may be based on a misunderstanding of what is complicated about task analysis. The actual methods used to analyze data are quite simple, and it is their baggage, much of it essential, unfortunately, that adds the complications. For example, one of the disadvantages of many of the task analysis CASE tools described in part IV is that (a) they demand inputs of certain sorts that may have to meet data completeness criteria that in a specific project situation are not relevant and (b) they are designed to produce very specific types of output. Chapter 4 also comments on the additional effort that a CASE tool may require (see also section 1.5.1.3).

None of the various chapters that purport to classify task analysis methods and CASE tools (e.g., chaps. 6 and 22–25) entirely succeed at providing a mechanism for selecting a particular method across all the situations in which task analysis might be applied in projects (which is not to say that they lack valuable information about the methods they cover). The general advice has to be to know the input data and the desired output and let these drive the selection of the analysis method. On balance, it is probably the required type of output that is the most important selection criterion. The reason for this is the lack of integration of many task analytic and software engineering methods (section 1.5.1.3).

As with many software engineering methods, in task analysis the representations used for analysis are usually the method's output representations, although some methods do have a translation stage so that they output a representation used in a software engineering method, such as a UML representation (section 1.5.1.3 and chaps. 7, 11, 12, 19, 22–24, and 27). Task analysis methods have stages and a small number of analysis operations within each stage. One aspect of HTA that makes it a simple task analysis method is that it has only a single main analysis representation. Most task analysis methods have a relatively small number of stages and representational forms, such as Scenario-Based Design (Carroll, 2000; chap. 5) and ConcorTask-Trees (chap. 24). Methods such as GOMS (chap. 4) and TAFEI (chap. 18) are partial task analysis methods in that the they deal with the later stages of what a task analyst does. TOOD (chap. 25) is an example of a complicated task analysis method that is probably only realistic to use, unless you are its inventors and intimates, when supported by a CASE tool.

Methods such as HTA are recognized as being difficult in that analysts need training and possibly years of experience to acquire expertise (chaps. 3 and 28). Indeed, many task analysis methods are rather messy in their description and in practice. In some cases, analysts must work around a method's different stages, although methods such as TOOD (chap. 25), control this rather better than most, partly as a consequence of its being supported by CASE tools. Diaper (2001b) discussed how building a CASE tool to support TAKD (the LUTAKD toolkit) made clear to the method's inventor the need to simplify the model of how analysts visit TAKD's stages. Feedback from task analysts trying to use TAKD indicated that stage navigation was a difficulty and, although it is only a small program, one of the LUTAKD toolkit's most important facilities was a navigation system that forced analysts to make very simple decisions concerning what analysis stage to do next.

Task analysis methods do up to three things with their input data: extract it, organize it, and describe it. Although task analysis data may come in many formats, the most common is a prose description (a "task transcript" or "scenario") of one or more tasks. Other formats, such as check sheets (Diaper, 1989a), interview notes, ethnography reports, and so forth, can be treated like prose task descriptions. The extraction, organization, and description processes vary widely across different methods, but however these are done, they tend to have the general features described below.

Extraction typically involves two operations: Representing each task as a sequence of short sentences, known as an activity list, a task protocol, or an interaction script (Carroll, 2000; Diaper, 2002b), and classifying things. As noted in section 1.2.1 and, for example, by Diaper (1989c), the things tend to be organized into objects and actions, and often the objects are separated into agents and things used by agents. Classifying things in other ways has developed over the last dozen years or so, as triggers, for example (chaps. 14 and 19).

Organization also involves two typical operations: integrating different task descriptions if more than one description of a task is used and categorizing the information extracted. The most common organization is a hierarchical one, although section 1.2.2.1 argues for heterarchical models, and some methods, such as simplified set theory for systems modelling (SST4SM) (Diaper, 2000a, 2001a), can use a flat, nonleveled model to produce temporary alternative classification schemes. The most common consequence of the categorization process is the division of task descriptions into subtasks.

Naturally, almost all task analysis methods claim to be able to combine descriptions of a task performed by different people in different ways. Quite a few methods are able to combine different tasks into a single task representation. On the other hand, there are some outstanding problems with combining even the same task performed in different ways. This is most obvious in task analyses that use high-fidelity data capture methods, usually video recording, to document a specific realistic task. Given more than one recording of the same task, the different ways it is carried out ideally need combining in a single task representation. Readers of this handbook and the other task analysis literature should be alert to the various ways that most task analysis methods fudge this issue. Diaper (2002b) described Carroll's (2000) method of noting alternatives at the bottom of the activity list as quite crude, but at least the issue is recognized by Carroll; it is not in some of this handbook's chapters. The most common fudge is to have a single ideal task description that limits the description of use to an optimized set of subtasks and operations. Although many methods use this approach, TAFEI (chap. 18) provides a particularly clear example because it explicitly exploits the approach to look for deviations from the ideal task model (errors). Nor is it clear that abstracted task models (section 1.4.2.3) help solve this problem. In many cases, to represent different tasks, analysts must employ craft skill, and chapter 30 argues that this is one problem area that ought to be recognized and dealt with.

The task analysis description operations are what primarily distinguish different task analysis methods. The variety of these operations is vast, and a great deal of this handbook is devoted to describing the operations used by the task analysis methods addressed. The description operations aid in further extraction (Carroll, 2000; chap. 5) and organization or add properties to objects and actions (e.g., those having to do with sequence or preconditions). The next section describes the typical first stages of task analysis before the analysts use whatever description processes their chosen method specifies.

1.5.2 The First Stages of Task Analysis

Like so much in life, preparation is the key to doing a good task analysis. The analysts should start by addressing the following five questions:

1. Which project stages will use task analysis?
2. What do you want to know from the task analysis?
3. What is the most appropriate task analysis output format?
4. What data can be collected?
5. What task analysis method?

To obtain reasonable answers to these questions, the analysts will have to have done most, if not all, of the intellectual preparation for the task analysis. Questions 1, 2, and 3 concern the role of the task analysis in the project. Answering questions 4 and 5 will require some basic background understanding of the tasks to be analyzed. What remains to be sorted out is the logistics, who will do what tasks when. The logistics may be trivial, as in the case of a small, well-integrated design team generating scenarios, but they can be very complicated, as when video recordings are to be made in the workplace. With experience, making video recordings of a wide variety of tasks in different environments is quite easy, but there is a lot of craft trickery in dealing with ambient light and sound, camera angles, and so on, beyond knowing how to operate the camera. Indeed, there is a lot of craft trickery in all data collection methods, which is one more argument for the suggestion that task analysis should be done by those with expertise in it (see also section 1.6).

The following subsections attempt to describe the first steps in nearly all task analyses. Different task analysis methods use different terminology, of course, and some of what is described below is skipped or is implicit in some methods. The description is independent of the data collection method, but the default method used is making video recordings of a range of tasks by a number of different people in their real work environment. This is a fairly high-fidelity data collection method (section 1.4.2.1), but the same issues arise with other methods such as interviews and even with low-fidelity task simulations such as scenarios (Diaper, 2002b).

1.5.2.1 Data Collection and Representation

By the time analysts are ready to start collecting data for a task analysis, they will have a systemic model of the work system and application domain (sections 1.2 and 1.3). It may only be in their minds, but they must have identified the things that will be recorded. This is obvious for video data but is even true, at the other end of the fidelity continuum, for scenario generation, for it is necessary to know, in general terms, what the scenarios are to be about. Note that the act of writing a scenario is equivalent to making a video recording of a task. The systemic task analysis (STA) approach introduced in section 1.5.3 argues that if the analysts are required to have a systemic model, they should start by making it explicit.

Task selection involves identifying the tasks that will be used to supply data (section 1.5.1.4). One needs to identify the tasks, their number, who will do them, and whether similar tasks are to be done more than once and by different people, for example. In general, the higher the fidelity of the data collection method, the more expensive, in all sorts of ways, the data collection exercise. It is obviously usually much easier to generate some more scenarios than to have to revisit a workplace to make more video recordings. One important decision to make concerns the level of detail of the data that will be collected, as too high a level will require further data collection and too low a level will be more expensive and may cause the analysts confusion (section 1.5.1.1).

The result of a task data collection exercise will be a set of records that describe how a set of tasks were performed. The records might be video- or audiotapes, written scenarios, ethnography reports, interview notes, or summaries of document sources such as training manuals. Interestingly, high- and low-fidelity data collection methods, such as video recording and scenario generation, respectively, result in records that more closely follow task structure than intermediate methods such as interviewing, even when the interviews are task performance orientated (e.g., Diaper, 1990b; chap. 16; but see sections 1.4.1.1 and 1.5.1.4).

A scenario as a story about use (Carroll, 2000; chap. 5) is a task transcript, that is, a prose description of task performance. One of the minor myths about task analysis that seems to have survived for decades is that if a transcript is lacking, creating one is the next step in the task analysis. One justification, more common in industry than academe, for producing a task transcript, say from a video recording, is that this is an administrative task that can be done by labor cheaper than a task analyst. An argument against this arrangement is that doing transcription is a good way for analysts to understand their data and their analysis. For video transcription, the rough estimate, on which there seems to be a consensus, is that transcription takes about 10 times longer than the recording. A better alternative when a transcript is not available is to take the records, whether interview notes, video or audio recordings, or whatever and immediately convert them into the activity list format that is central to virtually all task analysis methods. If one does have a task transcript, then producing an activity list is the next step.

1.5.2.2 Activity Lists and Things

An activity list, sometimes called a "task protocol" or an "interaction script" (Carroll, 2000, chap. 10), is a prose text description of a task that has the format of a list. Each item on the list represents a task step. The heuristic that this author has promulgated for years (Diaper, 1989c, 2001b, 2002b) is that each line of an activity list should have a single main agent that performs a main action that affects other things (i.e., objects and agents). Depending on the level of detail, there may be short sequences of subsidiary actions specified, such as those that never vary (e.g., pressing the [Return] key after a command line input), and the main action will often entail quite lengthy strings of task components (e.g., when entering text using a word processor). People do carry out tasks or subtasks in parallel (e.g., simultaneously browsing their PC's diary while arranging a meeting with someone on the phone), and the advice here is to treat the two related tasks separately in the activity list and represent each of the related task components on adjacent activity list lines. Switching between two or more unrelated tasks can be treated in the same way, or separate activity lists can be constructed, if the interleaved tasks don't affect each other. The end result of the activity list construction process should be an ordered list of the steps performed to carry out each task recorded.

It is important to note that, to follow the above heuristic, it is necessary to have identified the things that will be represented in the activity list and to at least have classified them as actions, agents, and other objects (see sections 1.2.1 and 1.5.2.1). One might chose to extend this classification while constructing the activity list (e.g., identifying triggers, as proposed in chap. 19) or this might be done at a later analysis stage.

Generally the analyst will want to give each line on the activity list a unique identifier, and the obvious thing to do, it seems to most people, is to number the activity list lines. If you do this starting 1, 2, 3, 4, . . . , then, in practice, you are almost bound to run into difficulties. As I've mentioned, after 20 years of doing task analysis I've still never got an activity list right first time and always have to return to it once I've done some further analysis. If I numbered the lines, I would need to renumber them each time I made a change. Even if software is being used that propagates changes to activity list line numbers throughout all the analyses, such changes still give analysts difficulty. After all, the point of identifying each activity list line is so that it can be easily referred to and found. When I number activity list lines these days, I prefer to use intervals of 100 or 1,000 so that I can add lines subsequently without renumbering.

With observational data collection methods, it is possible to collect time data. Most analog and digital video cameras allow the recording to be time stamped, usually to the nearest second. Recording task performance on paper and using a stopwatch is quite a difficult task for the analyst, and the timing is probably only reliable to within 3–5 seconds. Whenever possible I always try to use video rather than pen and paper. If collecting the time data is free, as with video recording, then it is still an issue whether to put it on the activity list. The temptation is to do it because the data are available, but my experience is that time stamping an activity list is a time-consuming business and should be done only if it is essential to what will be analyzed subsequently. It usually has to be done when efficiency issues are the primary concern of the task analysis. One advantage of using GOMS after producing an activity list is that GOMS provides estimates of the time it takes to perform task components (chap. 4), so time doesn't have to be recorded on the activity list. Nontheless, it is a good idea to check the GOMS estimates for a sample of tasks or subtasks against the actual durations recorded.

Many HCI tasks involve a dialogue between a small number of agents. Where there are two, three, or four agents (four is probably the upper limit), a good format for an activity list is to represent each agent in a separate column. This is easy with two agents, typically a user acting as the agent of the work system and a computer acting as the application domain's agent (sections 1.3.2 and 1.3.3). One advantage of this format is that the 3Ps (perceive, process, and perform) can be applied to each agent's column, similar to the cognitive model advocated by TOOD (chap. 25; see also chap. 28). Another advantage is that the format allows representation of behavior by one agent and different processing and responses by other agents. For example, in ATC, the ATCO may type an input to a flight strip control system while being observed by the flight chief sitting next to the ATCO; the computer system and the flight chief will obviously treat the ATCO's behavior differently.

An activity list describes a single instance of one or more tasks by listing what occurs in sequence, if not in time, although it may be more or less concrete (i.e., more or less detailed; section 1.4.2.3). The problems that remain concern how to deal with parallel tasks, how to deal with interrupted and interleaved tasks, and how to combine activity lists to produce generic task descriptions. These problems are properly part of the task analysis processes discussed in the next subsection, and some of them are illustrated in section 1.5.3.4 (see also chap. 30).

1.5.2.3 Analysis in Task Analysis

To start with a true but apocryphal story, something was obviously learned since the first British HCI conference in 1985, because the next year, at HCI'86, there was a stock answer to most questions from the audience: "Oh, we did a task analysis." In most cases, this meant that they had stood over the user's shoulder and watched for a few minutes. Although something may be better than nothing, they did have go and find a user, after all, this really is not good enough, and it gives task analysis a bad name. One option, perhaps understandably not often mentioned by those who produce articles and books on task analysis methods, including many of the coauthors of this handbook, is to settle for producing activity list descriptions and to decide that no further analysis needs to be done. Producing activity list descriptions of tasks really does force an understanding of the individual tasks (see also chap. 4). The question is, how much more do the subsequent analyses add? Sometimes a lot, of course, but there are cases, for example, in requirements analysis, where what is learned from activity list construction is the most useful contribution of the task analysis. In addition, sometimes when evaluating a prototype, for example, it is easy to spot task performance problems directly from individual activity lists.

See section 1.5.3.4 for a discussion of the general sorts of operations involved in different task analysis methods. As noted in section 1.5.2.2 and chapter 30, there are issues to be resolved concerning how to integrate task descriptions and deal with parallel, interleaved, and alternative versions of tasks.

1.5.3 Systemic Task Analysis

The discussion in section 1.5.2 is intended to apply to task analysis in general, even though the advice about activity lists is quite specific because some such representation is common in most task analysis methods, although formats vary. In contrast, this section illustrates with a microscopically small example how the first stages of a task analysis can be carried out. This requires a particular notation and method, and what will be introduced is a new task analysis method, currently dubbed systemic task analysis (STA). This method takes the ideas about systems and work performance introduced in sections 1.2 and 1.3 as its basis. A full description of the STA method is not provided here, but just the representations that STA uses at each of its initial main stages.

The main stages in STA, which all iterate in something like a classic waterfall-type model (section 1.4.2), are as follows:

1. Static systemic modeling.
2. Data collection.
3. Activity list construction.
4. Dynamic systemic modeling.
5. Simplified set theory for systems modeling (SST4SM).

1.5.3.1 Static Systemic Modeling and the Example Scenario

Section 1.5.2.1 argues that at the start of a task analysis exercise the analyst must have some form of systems model, however implicit, so STA makes a virtue of this by having its first step make such a model explicit. Using ATC as an example (see section 1.3.2), a systems model such as shown in Fig. 1.4 is the sort of thing that can be quickly obtained by looking around a control tower and interviewing a few people after perhaps having done some background reading. Such a system model identifies the things in the system (section 1.2.1), models both their conceptual and communicative relationships (sections 1.2.2.1 and 1.2.2.2, respectively), and identifies various goals that start to define the work to be achieved (section 1.3.2). It is a static model, however, in that it does not represent either time or sequence. For this reason and because there is no explicit representation of performance, it is not a task analytic model, although the analysts will probably run the model in their minds (section 1.2.2.2).

For the example, let us further assume as was the case with our Manchester Airport study (Section 1.3.2), that the primary interest is in a requirements capture exercise which is starting by modelling how the current paper flight strip system performs. The work, in the early 1990s, was carried out with about a ten year planning horizon for replacing the paper based system with an electronic flight strip system.

1.5.3.2 Data Collection

Having gained access to the Manchester ATC tower, not an easy thing, as there are major safety considerations, we set up a video camera to record the ATCO sitting in front of the flight strip system. Despite the cacophony of a real ATC tower, what the ATCO and the flight chief, who sits next to the ATCO, say is audible on the video recording (getting good sound required some careful, pre-researched placement of the camera microphone). We also had an extension feed to the video camera to record both sides of the radio telephone communication between the ATCO and the aircraft (also researched in advance). The video camera is left to run for an hour, and as expected the workload increases during the session. The ATCO is interviewed immediately after the session using a post -task walk-through. In this walk-through, the ATCO stops and starts the recording of the session and explains what is going on, either spontaneously or prompted by us. We record the walk-through by pointing the camera at the TV screen showing the original task video while the ATCO makes comments. The time stamp on the original video is easily readable on the walk-through recording, which is also time stamped. The walk-through of the 1-hour Manchester Airport task video takes about 3 hours, a typical amount of time.

1.5.3.3 Activity List Construction and Analysis

Whatever the format of the task data collected, an activity list is a fundamental (perhaps the fundamental) representation of a task in task analysis. Table 1.1 shows a fragment of an activity list that might have been created from the Manchester Airport videos. Producing it would involve first watching the task video and identifying the actions and other things with their times, then going through the walk-through video. Table 1.1 represents what an activity list might look like after several cycles of development during which the analyst has refined the descriptions of the task steps and added comments. The irregular activity list line numbering reflects this cyclical development (section 1.5.2.2). The activity list could be further refined, as discussed below.

The activity list Table 1.1 follows the systems model shown in Fig. 1.4 by separating the work system, whose primary agent is the ATCO, from the application domain, which is the flight strip rack. In a paper-based flight strip system, the flight strip rack is an object and not an agent, as the rack has no processes (sections 1.2.1 and 1.4.2.1 and chap. 26). The focus of the task analysis in Table 1.1 is on what the ATCO does, and the application domain descriptions are represented at a higher level and less completely than the descriptions of the ATCO's behavior.

At the first activity list line, 4800, the ATCO is in radio communication when the assistant places a new flight strip in the rack; the ATCO finishes this task at line 5700. At line 5800, the ATCO starts to scan the flight strip rack. The subject of the comment attached to this line (i.e., the trigger for this task) is the sort of thing an expert task analyst would have asked about during the walk-through. In a further refined activity list, it might be represented by its own activity list line specifying the ATCO's cognitive operations. Line 6100, for example, does indicate the cognitive rule the ATCO claims to use. Having decided to deal with a particular aircraft at line 6100, the ATCO then drops into a subtask concerning when the aircraft's flight strip arrived on the flight strip rack. The ATCO queries the flight chief about the flight strip before returning to the main current task of dealing with the selected aircraft at line 6700.

TABLE 1.1
Fragment of an ATC Activity List

ID	*Time*	*Work System*	*Application Domain*	*Comments*
...				
4800	14:24	Assistant places new flight strip for aircraft AC 234, cocked, on rack.	New flight strip AC234, cocked, on rack.	*... ATCO on radio to aircraft AC 207. Note input from the assistant system to the work system.*
...				
5500	15:08	ATCO ends radio dialogue to aircraft AC 207.	Radio turned off.	
5600	15:10	ATCO updates flight strip AC 207 with pen.		
5700	15:12		Flight strip AC207 updated.	
5800	15:15	ATCO scans rack of live flight strips.		*Cognitive trigger: "Do at end of each task and do often."*
5900	15:22	ATCO scans rack for new, cocked flight strips.		*Cognitive trigger: "Do after live flight strips and if no immediate tasks."*
6000	15:24	ATCO reads flight strip AC 234.		
6100		ATCO decides flight strip AC 234 needs dealing with as its arrival time is within 5 minutes.		*Aircraft AC 234 is on a diverted flight.*
6200	15:28	ATCO takes flight strip AC 234 off rack and holds it in hands.	Flight strip AC 234 removed from rack.	*Physical ATCO behavior.*
6210	15:28	ATCO doesn't know exactly when flight strip AC 234 arrived on rack.		*ATCO says this "only a mild concern" as knew he'd looked at the whole rack "3 or 4 minutes ago" (Actual = 5m:15s ago).*
6250	... 15:28	Flight Chief is looking at radar.	Radar display is normal.	*Flight Chief's current activity.*
6270	15:28	ATCO is looking at whether the Flight Chief is "free."		*This is a complex work- and social-based decision but done frequently.*
6300	15:30	ATCO shows flight strip AC 234 to Flight Chief.		*The flight strip is moved to be within easy reading distance for the Flight Chief while still being held by the ATCO.*
6301	15:30	ATCO says to Flight Chief, "When did this arrive?"		
6450	15:33	Flight Chief reads flight strip AC 234.		
6550	15:36	Flight Chief attempts to recall when last saw Assistant near rack.		*This is not an official Flight Chief responsibility.*
6650	15:40	Flight Chief says to ATCO, "About 5 minutes ago."		*This triggers the ATCO to return to dealing with the AC 234 flight strip.*
6700	15:43	ATCO focuses attention on rack.		
...				

The activity list in Table 1.1 could be more detailed or less detailed. For example, some activity list lines could be combined, such as 5500 and 5560; 5800–6000 and 6700; and 6200–6301 and 6450–6650. The level of detail chosen for Table 1.1 was primarily driven by the author's perception of its understandability by this chapter's readers, but it is the sort of level of description he has often found useful when analyzing important tasks. ATC is highly safety critical, so some depth of analysis is usually warranted.

Section 1.5.2.3 suggests that producing an activity list can sometimes be sufficient for some projects' purposes. To produce one such as in Table 1.1 does require the analyst to understand, quite thoroughly, each task analyzed. Given that the main project that the Manchester Airport study was to contribute to involved designing computer replacement systems for the paper-based flight strip rack, there are two major design issues that arise just from the activity list fragment in Table 1.1: (a) new flight strip arrival time information, and (b) ATCO and flight chief cooperative work and communication and their estimates of each other's current workload.

At line 6100, the ATCO notices that an aircraft is due to become "live" in a few minutes and realizes that he doesn't know when the flight strip appeared in the rack. With a paper system, an obvious solution to this problem would for the assistant to write on each flight strip the time it is placed on the rack. The ATCO expresses only "mild concern" at this problem, although we might consider looking in more detail at the difference between the ATCO's estimates of when the rack was last scanned and the actual time between scans (e.g., is the difference between "3 or 4 minutes" and 5:15 minutes significant?). With an electronic flight strip system, there are a range of design issues. Apart from merely recording the time a flight strip arrives on the rack, each flight strip could have one or more clocks or countdowns associated with it. For example, each flight strip could display when it was last modified, how long until the next action is due, and when it was last attended to. The latter could be achieved by selecting individual strips, whole blocks of them, or all strips after scanning them (see chap. 13). Any such design is likely to have consequences for cognitive actions like the triggers identified in lines 5800 and 5900. One measure of subsequent design success might be the reduced probability of the ATCO and flight chief needing to discuss such problems, particularly when they are both busy.

Sitting next to each other in the current ATC work system, the ATCO and the flight chief talk a lot, and when both are not busy, not all of the talk is narrowly task focused. Some is even social persiflage, which may be quite important, as the two have to work in a highly coordinated fashion, trusting each other. Note that these are important psychological things about these people (Section 1.2.1). Even assuming that a decision has been made to keep the ATCO and the flight chief next to each other in front of the electronic system, lines 6250 to 6650 still highlight a number of design issues. They are the sort of issues that task analysis tends to be better at identifying than other HCI and software engineering methods. First, the ATCO has to be able to estimate what the flight chief is doing, so he needs some way to get an overview quickly, and presumably the same is true for the flight chief. Second, the ATCO wants the flight chief to read a particular strip but does not want to give the flight chief control of it (information garnered from the post -task walk-through). These two issues create computer-generated display requirements concerning the general observability and specific readability of flight strips by someone not directly in front of the display. Interestingly, even after extensive task analyses, the actual new ATC system, described in chapter 13, had initial problems with display readability. A really creative task analyst might also foresee some design issues concerning less variety of style in both the display (e.g., the loss of variation in hand written things an flight strips when they are computer generated) and the ATCO and flight chief's behaviors, which will make estimating what the other person is doing harder in the new electronic system.

The above is a lot of design- and evaluation-relevant stuff to get from a few lines of a single task's activity list. Analyze more lines and more tasks and a great deal more design requirements can be taken directly from the task activity lists. Why do more analysis?

1.5.3.4 Abstract and Dynamic Systemic Modeling

Why do more task analysis after constructing an activity list? The most obvious answer has to do with the volume of information. In many, probably most, task analyses, the analyst winds up with many tens and often hundreds of pages of activity lists. What task analysis methods solve, more or less successfully, is the problem of reducing this volume by combining tasks, representing them at higher levels of description so as to allow different task instances to be combined, and representing alternative and optional tasks (e.g., an error-recovery subtask). Most task analysis methods do other things as well, and these things partially distinguish them from each other.

HTA is one of the hardest methods to do well. It is a popular method in industry but is not always done well (Ainsworth & Marshall, 1998) because it is poorly specified and relies on considerable craft expertise (chaps. 3 and 28). Although they look more complicated because they are better specified, particularly the task analysis methods supported by CASE tools, many methods are actually easier to carry out than HTA because the analyst is guided through the process (section 1.5.1.5). Notwithstanding my personal preference for heterarchical models (section 1.2.2.1), Fig. 1.5 shows where one might start producing a hierarchical model from the activity list fragment in Table 1.1. An inductive, bottom-up approach (section 1.5.1.4) has been used, and at this stage the specific activity list lines have been left in as hierarchy terminators below some initial task classifications. Whether the resulting hierarchy is actually an HTA diagram as described by Annett (chap. 3) depends on how these task classifications are defined; in an HTA "Deal with new flight strips," for example, would be an ATCO goal and not a description of behavior. As more activity list lines become incorporated in the hierarchical analysis, the model will change. For example, not all the subtasks that follow the "Identify

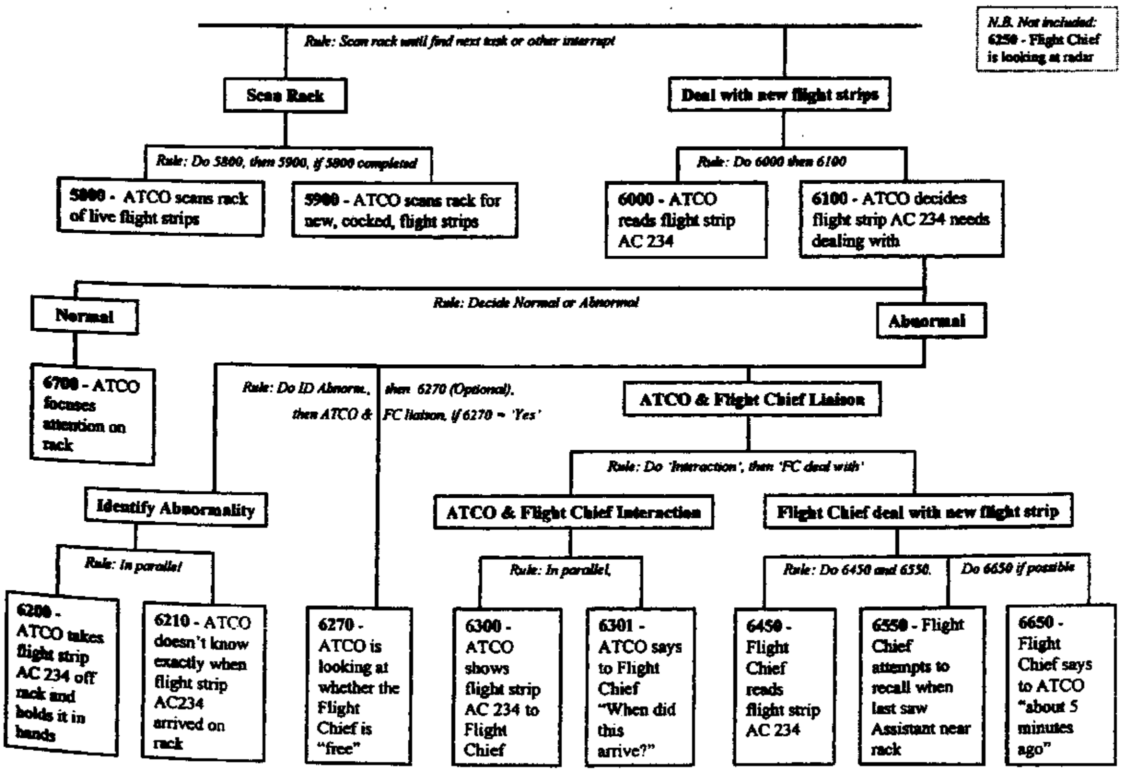
N.B. Not included:
6250 - Flight Chief is looking at radar
Rule: Scan rack until find next task or other interrupt
Scan Rack
Deal with new flight strips
Rule: Do 5800, then 5900, if 5800 completed
5800 - ATCO scans rack of live flight strips
5900 - ATCO scans rack for new, cocked, flight strips
Rule: Do 6000 then 6100
6000 - ATCO reads flight strip AC 234
6100 - ATCO decides flight strip AC 234 needs dealing with
Rule: Decide Normal or Abnormal
Normal
Abnormal
6700 - ATCO focuses attention on rack
Rule: Do ID Abnorm., then 6270 (Optional). then ATCO & FC liaison, if 6270 = 'Yes'
ATCO & Flight Chief Liaison
Rule: Do 'Interaction', then 'FC deal with'
Identify Abnormality
ATCO & Flight Chief Interaction
Flight Chief deal with new flight strip
Rule: In parallel
Rule: In parallel,
Rule: Do 6450 and 6550. Do 6650 if possible
6200 - ATCO takes flight strip AC 234 off rack and holds it in hands
6210 - ATCO doesn't know exactly when flight strip AC234 arrived on rack
6270 - ATCO is looking at whether the Flight Chief is "free"
6300 - ATCO shows flight strip AC 234 to Flight Chief
6301 - ATCO says to Flight Chief "When did this arrive?"
6450 - Flight Chief reads flight strip AC 234
6550 - Flight Chief attempts to recall when last saw Assistant near rack
6650 - Flight Chief says to ATCO "about 5 minutes ago"

FIG. 1.5. The start of an example hierarchical model based on the activity list in table 1.1.

abnormality" subtask will involve "ATCO & flight chief liaison," so incorporating more activity list lines will require some structural alteration to the diagram. One of the things that does drive this sort of analysis is that combining activity list lines generally forces abstraction (section 1.4.2.3). Obviously, if the comment for line 8000 is "ATCO reads flight strip AC321," which is very close to the comment for line 6000, then these two lines can simply be combined by deleting or making general the aircraft ID. In contrast, line 6100 might have a quite complex extension, with more activity list line analysis. For example, there may be alternative strategies to what "needs dealing with" means, using a variant like "needs dealing with soon" which might cause the ATCO to continue scanning the rack but to return to a strip as soon as possible after completing the scan.

One major difficulty that HTA shares with other task analysis methods and systemic models of all kinds is that, although a model can be invalid (i.e., not match what actually happens; sections 1.4.2.1, 1.5, and 1.5.1.4), in most situations there can be a number of different valid models. Making decisions on how to build a model is one of the core craft skills of virtually all analysts, whether in task analysis, IT systems analysis, software engineering, and so on.

STA attempts to deal with the above problem by providing a great deal of flexibility in the structure of its systems model. Starting with a systems model such as in Fig. 1.4, activity list lines can be added to the diagram by directly locating the lines (numbers) on things, including the communcative relationships. This is how the Pentanalysis Technique (Diaper et al., 1998) works, but there the systems model is a hierarchical DFD–based one rather than a heterarchical one. The systems model will frequently have to be refined, often by dividing things into smaller things. The end result of this analysis process in STA is a systems model with all the routes the different tasks take through it. The model thus combines the tasks and represents all the alternatives discovered. The remaining problem for STA, and a fundamental one for all HCI task analysis methods, is how to use the results of the analysis to support the project. Note that a good analyst would already have decided this (section 1.5.1.2).

STA uses a formal modeling system based on set theory called Simplified Set Theory for Systems Modeling (SST4SM). SST4SM is simplified in the sense that it is designed to be understandable, and even usable, by the "mathematically challenged" (Diaper, 2000a, 2001a). SST4SM's equations cause activity list lines to be rewritten in terms of the sets involved in the systems model and can have properties of each set or subset represented as set elements. With sensible set names, SST4SM's equations can be rewritten as English-like sentences, so the method provides one mechanism of automatic abstraction by producing prose activity list lines in a common format. This was a major, explicit design consideration for TAKD (e.g., Diaper 2001b; Diaper & Johnson, 1989). As briefly illustrated in Diaper (2000a), SST4SM has methods for understanding and reasoning with its equations that do not involve algebraic manipulation. The use of SST4SM is peculiar to STA, but the first stages of STA's approach would seem to cover what needs doing in virtually any task analysis exercise of quality. When a task analysis has not been performed well, a likely cause is that the analyst has not understood (and built) an adequate systems model, systematically collected the data, or adequately represented the data in some activity list form.

1.6 CONCLUSION

Is task analysis easy to understand? Based on the chapter's word count, you might be inclined to say no. On the other hand, although this chapter took more than half a year to write and contains a distillation of the author's 20 years of experience with task analysis, the concepts, each taken in turn, are not that complicated. Also note that the whole first half of the chapter is devoted to exposing, in a rational order, the many issues that underlie task analysis and which most writings on the subject address only partially if at all. The second half then tries to illustrate the practical aspects of the first half's theorizing. Section 1.1 describes the chapter's contents and so a summary will not be repeated here.

In a number of places in this chapter and in other chapters in this handbook, it is suggested that task analysis should be carried out by experts. There are two general arguments supporting this suggestion: (a) Task analysis is so complicated that only experts can understand it, and (b) like many things in life, doing task analysis well requires all sorts of craft skills that people really only learn through practical experience. Although agreeing with (b), this chapter attempts to refute (a). Although design is clearly a craft, chapter 11 makes explicit the value of using task representations that nonexperts can understand, and chapter 10 describes a commercial software development environment where task analysis is to be carried out by people who are not HCI experts.

Most of this handbook's chapters are cross-referenced in this chapter. Yet, because the field of task analysis is divers and fragmentary, as stated in the preface, the editors have refrained from standardizing the models, concepts, and terminology that occur in the other chapters. Readers may find that attempting to relate the contents of the other chapters to the ideas in this one is an informative exercise and will help elucidate what each of the other chapters actually covers. Whether editorial standardization might be enforceable in future editions of this handbook remains to be seen. As a prerequisite, people must agree on solutions to the problems of task analysis discussed in the final chapter of this handbook.

Finally, for those readers who are still struggling to understand the underlying concepts of task analysis, the next chapter provides a practical task analysis tutorial aimed to get students going at doing task analysis.

REFERENCES

Ainsworth, L., & Marshall, E. (1998). Issues of quality and practicality in task analysis: Preliminary results from two surveys. *Ergonomics, 41*, 1607–1617.

Alavi, M. (1993). An Assessment of electronic meeting systems in a corporate setting. In G. De Michelis, C. Simone, & K. Schmidt (Eds.), *Proceedings of the Third European Conference on Computer Supported Cooperative Work.*

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, R., Carroll, J., Grudin, J., McGrew, J., & Scapin, D. (1990). Task analysis: The oft missed step in the development of computer-human interfaces: Its desirable nature, value and role. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Human-computer Interaction: Interact, '90* (pp. 1051–1054). Amsterdam: North-Holland.

Annett, J., & Stanton, N. A. (Eds.) (1998). Editorial [Special issue on task analysis]. *Ergonomics, 41*, 1529–1536.

Ashworth, C., & Goodland, M. (1990). *SSADM: A practical approach.* New York: McGraw-Hill.

Atkinson, M. (1988). Cognitive science and philosophy of mind. In M. McTear (Ed.), *Understanding cognitive science* (pp. 46–68). Chichester, England: Ellis Horwood.

Barlow, J., Rada, R., & Diaper, D. (1989). Interacting WITH computers. *Interacting With Computers, 1*, 39–42.

Beer, M., & Diaper, D. (1991). *Reading and writing documents using Headed Record Expertext.* In M. Sharples (Ed.). *Proceedings of the Fourth Annual Conference on Computers and the Writing Process* (pp. 198–207). University of Sussex, Brigaton, UK. Reprinted in AISB Newsletter, (1991), *77*, 22–28.

Bell, J., & Hardiman, R. J. (1989). The third role: The naturalistic knowledge engineer. In D. Diaper (Ed.), *Knowledge elicitation: Principles, techniques and applications.* (pp. 47–86). Chichester, England: Ellis Horwood.

Bench-Capon, T., & McEnry, A. (1989a). People interact through computers not with them. *Interacting With Computers, 1*, 31–38.

Bench-Capon, T., & McEnry, A. (1989b). Modelling devices and modelling speakers. *Interacting With Computers, 1*, 220–224.

Benyon, D. (1992a). The role of task analysis in systems design. *Interacting With Computers, 4*, 102–123.

Benyon, D. (1992b). Task analysis and system design: The discipline of data. *Interacting With Computers, 4*, 246–259.

Benyon, D., & Macaulay, C. (2002). Scenarios and the HCI-SE design problem. *Interacting With Computers, 14*, 397–405.

Bowker, G. C., & Star, S. L. (2000). *Sorting things out: Classification and its consequences*. Cambridge, MA: MIT Press.

Cameron, J. R. (1983). *JSP & JSD: The Jackson approach to software development*. IEEE Computer Society Press, Los Angeles.

Carroll, J. M. (1991). History and hysteresis in theories and frameworks for HCI. In D. Diaper & N. Hammond, (Eds.), *People and computers VI* (pp. 47–56). Cambridge: Cambridge University Press.

Carroll, J. M. (2000). *Making use: Scenario-based design for human-computer interactions*. Cambridge, MA: MIT Press.

Checkland, P. (1981). *Systems thinking, systems practice*. New York: Wiley.

Clark, R. B., & Panchen, A. L. (1971). *Synopsis of animal classification*. London: Chapman & Hall.

De Marco, T. (1979). *Structured analysis and system specification*. Englewood Clifts NJ: Prentice-Hall.

Diaper, D. (1982). *Central backward masking and the two task paradigm*. Unpublished Ph.D. dissertation, University of Cambridge.

Diaper, D. (1984). An approach to IKBS development based on a review of "Conceptual Structures: Information Processing in Mind and Machine" by J. F. Sowa. *behavior and Information Technology, 3*, 249–255.

Diaper, D. (1986). *Identifying the knowledge requirements of an expert system's natural language processing interface*. In M. D. Harrison & A. F. Monk (Eds.), *People and computers: Designing for usability* (pp. 263–280). Cambridge: Cambridge University Press.

Diaper, D. (1989a). Task observation for Human-Computer Interaction. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 210–237). Chichester, England: Ellis Horwood.

Diaper, D. (1989b). Designing expert systems: From Dan to Beersheba. In D. Diaper (Ed.), *Knowledge elicitation: Principles, techniques and applications* (pp. 15–46). Chichester, England: Ellis Horwood.

Diaper, D. (1989c). Task Analysis for Knowledge Descriptions (TAKD): The method and an example. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 108–159). Chichester, England: Ellis Horwood.

Diaper, D. (1989d). The discipline of human-computer interaction. *Interacting With Computers, 1*, 3–5.

Diaper, D. (1989e). Giving HCI away. In A. Sutcliffe & L. Macaulay (Eds.), *People and computers V* (pp. 109–120). Cambridge: Cambridge University Press.

Diaper, D. (1990a). Simulation: A stepping stone between requirements and design. In A. Life, C. Narborough-Hall, & W. Hamilton (Eds.), *Simulation and the user interface* (pp. 59–75). London: Taylor & Francis.

Diaper, D. (1990b). Analysing focused interview data with Task Analysis for Knowledge Descriptions (TAKD). In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Human-computer interaction: Interact, '90* (pp. 277–282). North Holland: Elsevier.

Diaper, D. (1997). Integrating human-computer interaction and software engineering requirements analysis: A demonstration of task analysis supporting entity modelling. *SIGCHI Bulletin, 29*(1).

Diaper, D. (2000a). Hardening soft systems methodology. In S. McDonald, Y. Waern, & G. Cockton (Eds.), *People and computers XIV* (pp. 183–204). New York: Springer.

Diaper, D. (2001a). The model matters: Constructing and reasoning with heterarchical structural models. In G. Kadoda (Ed.), *Proceedings of the Psychology of Programming Interest Group 13th Annual Workshop* (pp. 191–206). Bournemouth: Bournemouth University.

Diaper, D. (2001b). Task Analysis for Knowledge Descriptions (TAKD): A requiem for a method. *behavior and Information Technology, 20*, 199–212.

Diaper, D. (2002a). Human-computer interaction. In R. B. Meyers (Ed.), *The encyclopedia of physical science and technology* (3rd ed.; vol. 7; pp. 393–400). New York: Academic Press.

Diaper, D. (2002b). Scenarios and task analysis. *Interacting With Computers, 14*, 379–395.

Diaper, D. (2002c). Task scenarios and thought. *Interacting With Computers, 14*, 629–638.

Diaper, D., & Addison, M. (1991). User modelling: The Task Oriented Modelling (TOM) approach to the designer's model. In D. Diaper & N. Hammond (Eds.), *People and computers VI* (pp. 387–402). Cambridge: Cambridge University Press.

Diaper, D., & Addison, M. (1992). Task analysis and systems analysis for software engineering. *Interacting With Computers, 4*, 124–139.

Diaper, D., & Johnson, P. (1989). Task Analysis for Knowledge Descriptions: Theory and application in training. In J. Long & A. Whitefield (Eds.), *Cognitive ergonomics and human-computer interaction* (pp. 191–224). Cambridge: Cambridge University Press.

Diaper, D., & Kadoda, G. (1999). The process perspective. In L. Brooks, & C. Kimble (Eds.), *UK Academy for Information Systems 1999 Conference Proceedings* (pp. 31–40). New York: McGraw-Hill.

Diaper, D., McKearney, S., & Hurne, J. (1998). Integrating task and data flow analyses using the Pentanalysis Technique. *Ergonomics, 41*, 1553–1583.

Diaper, D., & Shelton, T. (1987). Natural language requirements for expert system naive users. In *Recent developments and applications of natural language understanding* (pp. 113–124). London: Unicom Seminars Ltd.

Diaper, D., & Shelton, T. (1989). Dialogues With the Tin Man: Computing a natural language grammar for expert system naive users. In J. Peckham (Ed.), *Recent developments and applications of natural language processing* (pp. 98–116). London: Kogan Page.

Diaper, D., & Waelend, P. (2000). World Wide Web working whilst ignoring graphics: Good news for web page designers. *Interacting With Computers, 13*, 163–181.

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

Downs, E., Clare, P., & Coe, I. (1988). *Structured systems analysis and design method: Application and context*. Englewood Cliffs, NJ: Prentice Hall.

Easterbrook, S. (Ed.). (1993). *CSCW: Cooperation or conflict?* New York: Springer-Verlag.

Eva, M. (1994). *SSADM version 4: A user's guide* (2nd ed.). New York: McGraw-Hill.

Green, T. R. G. (1990). The cognitive dimension of viscosity: A sticky problem for HCI. In D. Diaper, D. Gilmore, G. Cockton, & B. Shackel (Eds.), *Human-computer interaction: interact '90*. (pp. 79–86). Amsterdam: North-Holland.

Hammer, M. (1984, February). The OA mirage. *Datamation*, pp. 36-46.

Hares, J. (1990). *SSADM for the advanced practitioner*. New York: Wiley.

Harrison, M., & Thimbleby, H. (1990). *Formal methods in human-computer interaction*. Cambridge: Cambridge University Press.

Hinton, G. E., & Anderson, J. A. (Eds.). (1981). *Parallel models of associative memory*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Johnson, P., Diaper, D., & Long, J. (1984). *Tasks, skills and knowledge: Task analysis for knowledge based descriptions*. In B. Shackel (Ed.), *Interact '84:* First IFIP Conference on Human-Computer Interaction (pp. 23–27). Amsterdam: North Holland.

Life, A., Narborough-Hall, C., & Hamilton, W. (1990). *Simulation and the user interface*. London: Taylor & Francis.

Lim, K. Y., & Long, J. (1994). *The MUSE method for usability engineering*. Cambridge: Cambridge University Press.

Long, J. (1986). People and computers: Designing for usability. In M. Harrison, & A. Monk (Eds.), *People and computers: Designing for usability* (pp. 3–23). Cambridge: Cambridge University Press.

Long, J. (1997). *Research and the design of human-computer interactions or "What Happened to Validation?"* In H. Thimbleby, B. O'Conaill, & P. Thomas (Eds.), *People and computers XII* (pp. 223–243). New York: Springer.

Long, J., & Dowell, J. (1989). Conceptions of the discipline of HCI: Craft, applied science, and engineering. In A. Sutcliffe & L. Macaulay, (Eds.), *People and computers V* (pp. 9–34). Cambridge: Cambridge University Press.

Lowe, E. J. (2002). *A survey of metaphysics*. Oxford: Oxford University Press.

Mumford, E. (1983). *Designing participatively*. Manchester, England: Manchester Business School Press.

Norman, D. (1986). Cognitive engineering. In D. Norman, & S. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 31–61). Hillsdale NJ: Lawrence Erlbaum Associates.

Nussbaum, M. C. (2001). *Upheavals of thought: The intelligence of emotions*. Cambridge: Cambridge University Press.

Patching, D. (1990). *Practical soft systems analysis*. London: Pitman.

Pressman, R. S. (1994). *Software engineering: A practitioner's approach* (European ed.). London: McGraw-Hill.

Pullinger, D. J. (1989). Moral judgements in designing better systems. *Interacting With Computers, 1*, 93–104.

Ross, K. A., & Wright, C. R. B. (1988). *Discrete mathematics* (2nd ed.). Englewood Cliffs NJ: Prentice-Hall.

Shapiro, D., & Traunmuller, R. (1993). CSCW in public administration: A review. In H. E. G. Bonin (Ed.), *Systems engineering in public administration* (pp. 1–17). New York: Elsevier.

Sharples, M. (1993). A study of breakdowns and repairs in a computer-mediated communication system. *Interacting With Computers, 5*, 61–78.

Shepherd, A. (2001). *Hierarchical task analysis*. London: Taylor & Francis.

Sommerville, I. (1989). *Software engineering*. (3rd ed.). Reading, MA: Addison-Wesley.

Sutcliffe, A. (1988). *Human-computer interface design*. London: Macmillan.

Watling, J. L. (1964). Descartes. In D. J. O'Connor, (Ed.), *A critical history of western philosophy* (pp. 170–186). London: Macmillan.

Zhang, P. (1999). Will you use animation on your web pages? In F. Sudweeks, & C. T. Romm (Eds.), *Doing business on the Internet: Opportunities and pitfalls* (pp. 35–51). New York: Springer.