More than we think:
Facing up to the plurality of goals, methods, needs, and resources in HCI

by
Stephen W. Draper
GIST (Glasgow Interactive Systems cenTre)
University of Glasgow
Glasgow G12 8QQ U.K.
email: steve@psy.gla.ac.uk
WWW URL: http://www.psy.gla.ac.uk/~steve

Published: Australian Journal of Information Systems (1996) vol.3 no.2 pp.31-35.

Keywords
Affordance, activity theory, plurality, tasks, guessability, learnability, experienced user performance, learning by exploration.

Abstract
Most analyses of how humans use artifacts, and interactive software in particular, have a strong tendency to assign 1:1 correspondences between goals and methods: to see software as supporting one task, users as having one way of executing a task, one thing to learn when learning a command, and one source for discovering the information. In fact this is a rare case, and multiplicity of goals, methods, information needs, and information resources is the rule even in simple software. How this causes problems for the design and testing of user interfaces can be illustrated by examples from a wide range of domains and levels of design, including studies on learning by exploration, the effect of machine delays on user strategies, the learnability of icon sets, evaluation studies of Computer Assisted Learning, and an analysis of the concept of affordance. Such plurality can be a source of robustness for the performance of interfaces: it is a problem mainly for analysis and HCI research, which struggle to account for the frequent case of high average performance levels mixed with a few residual problems. To address this plurality, we must extend our analyses to cover sets of alternative methods for tasks rather than single user procedures, and perhaps draw on concepts such as Activity Theory to address users' mental organisation of such plurality.

Introduction
We can detect and correct the main bugs in user interfaces by observing users having problems and employing an iterative design and testing cycle, just as in practice we find the main bugs in software by the initial informal testing of "trying it out". What we cannot do is approach a guarantee of bug free user interfaces through comprehensive testing, as we would for non-interactive software. Proving correctness through exhaustive testing of all possible inputs (black box testing) is usually not possible because of the combinatorial explosion of possible inputs, and this applies also to HCI since the "inputs" here would include all possible prior states of knowledge of the user. Software engineers therefore often use "white box testing", where they use their knowledge of the internal structure of the code to exercise every branch of it. In HCI we have to test the combined system of user and machine, so white box testing would require knowing what "procedures" or "methods" the user is executing. Although task analysis purports to do this, a growing range of observations show that users are much more variable in their methods than analysts assume, and indeed more variable then they are aware of themselves.

The rational way to design software is usually to organise the code so that each function needed as part of the design is implemented once and once only. On the one hand functions are not duplicated wastefully (providing two ways to do one job), but on the other hand each function usually does only one job so that if modification is needed for one task, others are not disrupted and need not be considered during the redesign. This approach of having a 1:1 correspondence between jobs and methods saves code (comparable to a person economising on learning), maximises re-use, and is thought to raise reliability over the program's life.

Most analyses of how humans use artifacts, and interactive software in particular, have a strong tendency to follow this approach and assign 1:1 correspondences between goals and methods: to see software as supporting one task, users as having one way of executing a task, one thing to learn when learning a command, and one source for discovering the information. However it seems humans are not like that. They do not usually all follow a single method, nor memorise the same things about an interface, nor follow the same goals. Multiplicity of goals, methods, information needs, and information resources is the rule even in simple software. How this causes problems for studies of user behaviour can be illustrated by a wide range of examples.

Tasks

In many cases, interactive software and its user interface has to support multiple tasks with different requirements. Sørgaard [8] discusses designs for a seat reservation system for railways. He criticises one based on traditional database queries which only supports requests for seats in terms of a few salient, explicitly named attributes e.g. window or aisle, facing forward or back, and proposes one that shows the seating plan for the whole train in a diagram that both customer and clerk can see and point to. This design allows customers to point to the seats they prefer, and allows them to take many possible attributes and relationships into account e.g. distance from the dining car, distance from the door, etc. Sørgaard's design finesses any laborious attempt to discover all the "tasks" customers are trying to perform, and yet will support a far greater range of seat specification types including those that might be very rare in the user population as a whole. It seems clearly superior exactly because it will support tasks the designer could not have anticipated. The general point therefore seem to be that designers cannot foresee the tasks (top level goals) that users will in fact bring to artifacts. Another example commonly quoted is that of spreadsheets, which are said to have been designed originally to support accounting (calculating totals of money flows), but are now widely, even predominantly, used to support financial planning in which the bottom line is entered as a given and possible component values calculated backwards from that. The way in which a new artifact often generates new, unforeseen, applications is also often commented upon, particularly by task-artifact cycle theorists (e.g. [1]), who have developed a well elaborated account of this phenomenon.

Sørgaard's design suggests that there may be ways for designers to cope with this, but not by relying on task analysis in any usual form. Instead, it is a case where the task analysis needed is not of specific individual tasks (which it may not be practicable to ennumerate), but of sets of tasks. It is not an accident that the solution offered is in the form of a particular visual representation to be displayed, and indeed to form the centre of the user interface. Most really successful interfaces seem to be based around such a representation, whose virtue is to serve multiple implicit needs well.

However the need to support multiple tasks with different requirements also occurs at a lower level, and in almost every program. A study [3] of how users approach learning a new application program (MacPaint) showed that roughly half their time was spent in learning by exploration (LBE) - not in searching for a command to achieve a concrete task but in using a command experimentally to learn its effect. Few if any task analyses have learning as a task, and so cannot capture this important and frequent user behaviour. Furthermore the requirements for supporting LBE are different from those for supporting the material tasks associated with the same command. (The former requires action -> task mappings to be perceptible after the event, whereas material goals require task -> action mappings to be made by the user before the event.)

Whether subjects are instructed to explore or to do concrete tasks, both kinds of activity are seen: instruction seems to reverse the approximately 60:40 ratio of those activities, rather than abolishing either. In a few cases in some interfaces, even experienced users never learn to distinguish some commands and persist in rediscovering them by exploration as a permanent alternative to making the mnemonic effort.

Nevertheless, LBE is more important to first time users, and is a reflection of how the tasks as well as the methods users employ shift as they learn an interface. Hence users' needs in part depend on their experience with the interface, and can be divided into guessability (first time use), learnability, and EUP (experienced user performance). An interface must usually be designed (and tested) to satisfy all of these different needs.

Another general issue is the methods a user must discover for recovering from errors. In principle,there is one of these for every possible state of the machine. They may be infrequently executed, but are part of the set of user tasks a design must support, even though often forgotten in analyses.

Methods

Users exhibit plurality in methods (i.e. action sequences or procedures) as well as goals. At the highest level of tasks, software is just one means to some external end, as appears in the problem of evaluating the effectiveness of CAL (computer assisted learning) programs [4]. Here success at the task (i.e. learning) depends on the whole set of learning resources not just the software so that changing the software may have little measurable effect. For instance, if the CAL is defective on some point, students motivated to pass exams will simply switch to an alternative source such as the textbook. Consequently, little change in test results can be expected in real classroom settings as the quality of the CAL varies over a wide range. Artificial experiments can isolate the effect, but these are grossly unrealistic as they mean cutting students off from the multiple sources normally available e.g. help from teachers, consulting books. Thus when computers are considered in their work settings, they have to be considered as part of an ensemble of resources and methods. In this context lack of use is not necessarily a failure, any more than a trouser belt starts to "fail" when the wearer lies down and it ceases to be the factor holding the trousers in position.

At a lower level of task, that of learning basic command functions in software, users also normally employ diverse methods. Even in the first few hours of organised instruction, a group of students can be seen to develop diverse methods for basic tasks. For instance, some students will use the Save command frequently (to back up the current version to disk), but others may wait to the end of their job and then use the Quit command which then leads them via a prompt into doing a save. Logfiles might suggest that some students never learned to save as they never use the Save command, but in fact they have just learned an alternative method for that task. In fact in a lot of software it is hard to find any function that can only be done in one way (e.g. to change the font of some text you can either type the text, select it, use a font command; or else select an insert position, use the font command, and type in the text); and this can lead to numbers of commands being apparently unused by some users.

In a study centering on the task of copy typing three digit numbers over periods long enough for users to reach "expert" level, three user procedures are seen. They differ in whether the users bother to look at the screen for the prompt, simply type ahead, or pause until they estimate the machine is ready again. An early study [11] argued that the machine response delay determined which was adopted, but our followup studies [7] showed more complex results, indictating that we cannot yet predict what method users will select, even when well practiced at the task. The order of training, the need to recover from errors, or just lapses in concentration may all cause switches in procedure.

Just as users exhibit diverse methods of action, so they exhibit diverse methods of recognising icons. Icons could be recognised by various features e.g. shape, position, and the question is which are actually used [6]. While some generalisations hold, users may in fact use different features of each icon in a set e.g. position for one, colour for another. There is evidence that the features remembered and used change with experience, although this cannot be detected by performance measures of time and errors: an example of a silent shift of cognitive method that both user and designer may be unaware of. Cf. [5].

Information needs and sources

The changes in icon feature used in recognition are related to a user's changing needs as they learn. Another set of needs can be grouped under the heading of "affordance" [3]. In general there are three different pieces of knowledge a user learns or otherwise gains access to for each command, connecting it to action knowledge at its own level and at the levels above and below. Consider for example the command "paste" in Macintosh applications. a) It copies text from a hidden buffer into the current document (what the command does at its own level). Users have to learn that the name "paste" refers to this action, and conversely that when they require that effect they should search for that command name. b) It is typically used as part of a standard two step plan of cut then paste to move text (how it fits into action plans and tasks at the next level up). This is almost impossible for users to discover for themselves (searching for a "move" command will not help), and they probably learn this from other users talking about "cut and paste". c) To execute it the user must first use the mouse to select the insertion point and then use the "edit" menu (the action level below). These three kinds of affordance information compete for expression in the interface design e.g. in the choice of command name. For instance type (c) how-to information is usually supplied by using a single consistent visual appearance for buttons and menus, so the user will try to operate any item of that kind; whereas if menus are concealed then users may not discover them at all. For instance, in THINK Pascal on the Macintosh, the window's title is really a pull down menu if you hold down the command key, but users never discover this without being told. Of course, the title bar could have had an instruction ("click on me"). To express type (b) information about higher level tasks, a designer can offer a menu command named for the task that then leads the user through the necessary sequence of actions: "Print" is of this kind, and similarly a "Move" command could be offered like that. Because of this competition for expression, many items of information remain unexpressed: an example of the plurality of user information needs lying behind the design of command presentation. Designers can seldom even enumerate these needs, much less adequately serve them all.

On the other hand, there is also a plurality of information sources. For instance users can learn either from the interface directly e.g. through self-explanatory command names, or by exploration. As long as one source works, the others may fail with little deterioration of performance. Hence an important issue in usabilty testing is how to decide if a momentary problem is in need of fixing. When "failures" in fact lead soon to a user trying an alternative action and succeeding this can be a successful example of LBE, which is a legitimate and important technique for communicating information to users. Conversely, however, the fact that users can often find alternative methods may conceal failures that perhaps should be fixed. The fact that students can learn from a textbook does not mean that ineffective CAL material should be left in use, and the fact that users can rediscover a command's meaning by exploration does not mean that an unlearnable or misleading command name or icon should be left in place.

Analysing plurality

Thus the pervasiveness of plurality significantly complicates rational testing, although it explains why interfaces with many imperfections can nevertheless produce good user performances on average (i.e. ignoring the minority of users and occasions where all methods fail together) thus reducing the urgency of perfecting them. We need to practice analysing, not the single way a user is expected to do something, but the complete set of ways they might try to do it. We need to begin to formulate desirable properties of these sets e.g. rapid convergence on task achievement, alternative methods that do not require the use of the eyes, or one hand, or prior expert knowledge, etc. And we need to formulate the need for a smooth learning path between one method and another (E.g. see [2] for how better to support the learning of keyboard accelerators for menu commands.)

One theory that may help us to understand at least some aspects of this pluralism underlying human behaviour and knowledge is Activity Theory (AT), not least because of its linking of consciousness to levels of behaviour. According to AT, our consciousness is focussed on the central level of actions — an analysis consistent with ideas of direct manipulation as engaging users by sustaining the feeling of doing things "directly". It is obvious that more automatic acts (e.g. positioning finger tips over keys) are largely beneath the level we are attending to and so unconscious or at least unattended to, but we should note that AT also implies that the level of activities above conscious actions is also in effect largely outside our awareness. We remain engaged with moment to moment actions rather than reflecting how these are or are not contributing to our larger goals. This is consistent with how people can lose track of time while working at direct manipulation tasks, and with how people are bad at giving an accurate account of how they do their work, as Suchman has argued for a long time [9, 10]. The frequent failure in HCI to attend to the need for LBE and for error recovery may be an effect of this: if you observe users you see them doing both, but if you ask them what they do when they use a piece of software they are unlikely to mention them. Thus partly because our attention both as users and as designers is usually concentrated on only one level, human behaviour and the properties of successful user interface designs have more complexity than we think.

References

[1] Carroll J.M., Kellogg, W.A. & Rosson, M.B. (1991) "The task-artifact cycle" in J.M.Carroll (ed.) Designing Interaction: psychology at the human-computer interface pp.74-102 (Cambridge, UK.: Cambridge University press).

[2] Dix,A. (1995) "Accelerators and toolbars: learning from the menu" pp.138-143. HCI'95 adjunct proceedings (eds.) G.Allen, J.Wilkinson, P.Wright (School of Computing & Mathematics, University of Huddersfield).

[3] Draper, S.W., & Barton, S.B. (1993) "Learning by exploration, and affordance bugs" pp.75-76 Interchi'93 Adjunct proceedings (eds.) S.Ashlund, K.Mullet, A.Henderson, E.Hollnagel, T.White (ACM)

[4] Draper,S.W., Brown,M.I., Edgerton,E., Henderson,F.P., McAteer,E., Smith,E.D., & Watt,H.D. (1994) Observing and measuring the performance of educational technology TILT project, Robert Clark Centre, University of Glasgow

[5] Kaptelinin, V. (1993) "Item recognition in menu selection: the effect of practice" pp.183-184 Interchi'93 Adjunct proceedings (eds.) S.Ashlund, K.Mullet, A.Henderson, E.Hollnagel, T.White (ACM)

[6] Moyes,J. (1995) Putting icons into context: the influence of contextual cues on the usability of icons PhD dissertation, dept. of Computing Science, University of Glasgow.

[7] O'Donnell,P.J. & Draper,S.W. (1995) "How machine delays change user strategies" pp.111-117. HCI'95 adjunct proceedings (eds.) G.Allen, J.Wilkinson, P.Wright (School of Computing & Mathematics, University of Huddersfield).

[8] Sørgaard, Pål (1988) A discussion of computer supported cooperative work Ph.D. thesis, Aarhus University, Denmark

[9] Suchman, L.A. (1983) "Office procedures as practical action: models of work and system design" ACM transactions on office information systems vol.1 pp.320-328.

[10] Suchman, L.A. (1995) "Making work visible" CACM vol.38 no.1 pp.56-64.

[11] Teal, S.L. & Rudnicky, A.I. (1992) "A performance model of system delay and user strategy selection" *Proc. CHI '92* pp.295-305