Analysis and simulation of user interfaces

Harold Thimbleby

Middlesex University, LONDON, N11 2NQ.

By taking a mobile phone as a worked example, we show how it and new interfaces can be simulated and analysed. A new interface is shown to reduce the optimal key press costs of accessing the phone's functionality, without losing usability benefits — this is a specific contribution to menu design. However, the approach is not limited to mobile phones, nor just to menus; the techniques are general and can be applied widely. A distinctive feature of the approach is that it is fully inspectable and replicatable — this is a contribution to the field of HCI more generally.

Keywords: user interface design, formal methods, menus, new user interfaces

1 Introduction

The analysis of user interfaces has largely concentrated on issues of human performance, behaviour and cognition. In comparison, device-oriented analyses of user interfaces are rare, which is strange because devices — unlike humans — are precisely known. In the design process, devices themselves are the main areas where usability improvements can be effected. This paper exhibits a range of user interface analyses from a device perspective. An actual device, in commercial production, is used as a case study, and we exhibit a functionally equivalent user interface that requires a third fewer keypresses to use than the original design on average, and whose worst case cost is just one sixth. The analyses of both the original and alternative user interfaces are described in sufficient detail to be replicated by other usability engineers.

1.1 Contributions to HCI

Sometimes papers in HCI describe ideas that are not replicable; often the systems described are inaccessible, obsolete or proprietary, or the experimental details are not described in sufficient detail, or the methodology used allows vagueness. Unspecified craft knowledge is often required to use methods reliably. Although this paper is part of a larger project, the work is fully replicable: all claims and results here can be reproduced. In any method, there is room for mistakes and confusion, and often they can go unnoticed — or may be concealed, accidentally or even deliberately; the approach here can be tested by the investigator or tested by others. Indeed, there are several ways to calculate all results claimed, and this provides additional checks and safeguards — in contrast to more approximate or descriptive approaches. As Richard Feynman put it, if there is something slightly wrong with our definitions or theories, then the mathematical rigour will convert these into ridiculous conclusions (many of which can be spotted automatically), which we will interpret and so correct. Indeed, automatic and other checks on the mathematics in this paper helped fix typos, most of them of the sort that would easily have been missed in less formal approaches.

A companion paper is available on the World Wide Web, which provides *all* information behind the results reported here. The method is straight-forwardly mathematical, which means there are many textbooks and other sources of information about it. But, further, the mathematics is 'packaged' as a program, on the web site, and the benefits claimed can be achieved without delving into the technical details. For example, all the diagrams and results shown in this paper were calculated from a single specification of an interactive device (which is included as an appendix to this paper).

The techniques for analysis used here can be used with other device specifications, merely by changing the appendix, or they can be developed for other purposes.

To prove that our approach can handle real designs, we start from an analysis of an accurate model of the menu user interface of the Nokia 5110 mobile phone. The general approach to design taken here could be used with any push button device, and would be particularly easy to employ when working within a design process that specifies the feature set of a device. (If we had worked with Nokia, of course we could have avoided reverse engineering the device's user interface, since the user interface specification should have anyway been directly available.)

1.2 Background

Practically, the present paper is a continuation of research going back to Hyperdoc (Thimbleby, 1993), which was a system for simulating and analysing user interfaces to simple push button devices. Hyperdoc was criticised for only handling small devices (Dix *et al*, 1998), and it was also a platform-dependent tool — it ran in HyperCard on the Apple Macintosh — and its inner workings were never published. In contrast the present approach is based in *Mathematica* (Wolfram, 1996), which is platform-independent, and permits the entire approach, including all its details, to be published. (Actually there is no need to use *Mathematica* — Java, for example, could have been used instead; but *Mathematica* happens to be much better documented than Java.)

Theoretically, the motivation behind this paper is expressed in (Thimbleby, 1994), and is also illustrated in approaches such as Furnas (1997). Our approach is in contrast to that actually used by Nokia (Väänänen-Vainio-Mattila & Ruuska, 2000).

We use *Mathematica* for the mathematical calculations — using good tools helps offset the additional cost of presenting results rigorously. *Mathematica* allows the user interfaces studied to be simulated, checked or have conventional usability experiments run on them; *Mathematica* can also generate specifications of the user interface that can be used by, say, Java or C programs, or even converted to hardware. Other advantages of *Mathematica* for HCI work are discussed elsewhere (Thimbleby, 1999), which further suggests how user manuals and other material can also be handled.

Mathematica is a cross between a word processor, graphics program and a symbolic mathematics tool. Like a word processor outliner, sections can be opened or closed to reveal different degrees of detail as needed. What is printed in these proceedings is only part of what the paper actually contains. For example, the *Mathematica* instructions to draw the figures are not needed for most readers of the printed paper, and are therefore concealed; however, the code is still 'inside' the original version of the paper. For example, in the full paper just before all Figures there is a piece of *Mathematica* code that generated and either plotted or typeset the Figures. To guarantee accuracy, all versions of the paper were generated automatically from a single master copy (though errors may have crept in during the printing process for the conference proceedings, which was beyond our control). In short, this paper and its illustrations were not created by a conventional error-prone 'cut and paste' approach.

2 The Nokia 5110 user interface

As a concrete case study, we will be concerned with the Nokia 5110 mobile handset's menu functions, though there are a number of essential functions that are not in the menu (such as quick alert settings and keypad lock). There are 84 features accessible through the menu. A softkey, labelled '–,' called *Navi* by Nokia, selects menu items; keys $\wedge$ and $\vee$ move up and down within menus. The correction key C takes the user up one level of the menu hierarchy, whose structure is illustrated in Figure 1. With reference to Figure 1, the function *Service nos* can be accessed from *Standby* by pressing Navi [the phone now shows *Phone book*], then pressing Navi [shows *Search*], then pressing $\vee$ [shows *Service Nos*] followed by a final press of Navi to access the function itself. All menu items have a numeric code (displayed on the Nokia's LCD panel); for example, *Service nos* can also be access by pressing Navi 1 2 (no final press of Navi is required).

There are some complications, which we ignore in this paper — they are also ignored in Nokia's user manual. For example, inconsistently, the C key does not work when shortcuts are being used, so Navi 2 C 1 is equivalent to Navi 2 1, not Navi 1. The Nokia 'completes' shortcuts, so that Navi 1 7 in fact selects *Type of view*, not *Options* (see Figure 1). There is no fixed relation between shortcuts and the position of functions in the menu, since some functions may not be supported (e.g., by particular phone operators): if *Service nos* is not available, pressing ∨ would move from *Search* directly to *Add entry*, but the shortcut for *Add entry* would still be Navi 1 3 (trying Navi 1 2 would get an error).

There is some ambiguity on what should be taken as a basic function, and what as an option within a function. For example, *Type of view* is treated by the User Manual as a function, but it has a submenu (*Name list*, *Name number*, *Large font*). For our definitive list, see the specification of the Nokia in Appendix 3, which was used to generate all the figures and graphs in the paper. It can easily be edited to do analyses based on any variations.

Phonebook -1
- **Search** -1-1
- **Service nos** -1-2
- **Add entry** -1-3
- **Erase** -1-4
- **Edit** -1-5
- **Send entry** -1-6
- Options -1-7
  - **Type of view** -1-7-1
  - **Memory status** -1-7-2
- **Speed dials** -1-8

Messages -2
- **Inbox** -2-1
- **Outbox** -2-2
- **Write messages** -2-3
- Messagesettings -2-4
  - Set1 -2-4-1
    - **Message centre number** -2-4-1-1
    - **Messages sent as** -2-4-1-2
    - **Message validity** -2-4-1-3

.....

Tones -9
- **Incoming call alert** -9-1
- **Ringing tone** -9-2
- **Ringing volume** -9-3
- **Message alert tone** -9-4
- **Keypad tones** -9-5
- **Warning and game tones** -9-6
- **Vibrating alert** -9-7

Figure 1. Extracts from the Nokia 5110's menu structure.

Figure 2 shows the Nokia's *Standby* function at the top, and each horizontal row downwards is a group of functions that each take an equal minimum number of key presses to access from *Standby* (ignoring the numeric shortcuts). Of the 188 circles, 84 circles are black: these indicate phone functions of actual use, as opposed to submenus that in themselves have no other purpose than structuring the user interface, such as *Options*.

Because of the layout of Figure 2, Navi (which selects items from menus,whether submenus or functions) moves downwards, and C (which corrects errors) goes upwards; the Up and Down keys do not move in a systematic direction in this layout. Thus the Figure shows the minimum costs of accessing functions, rather than the menu hierarchy (as in Figure 1).

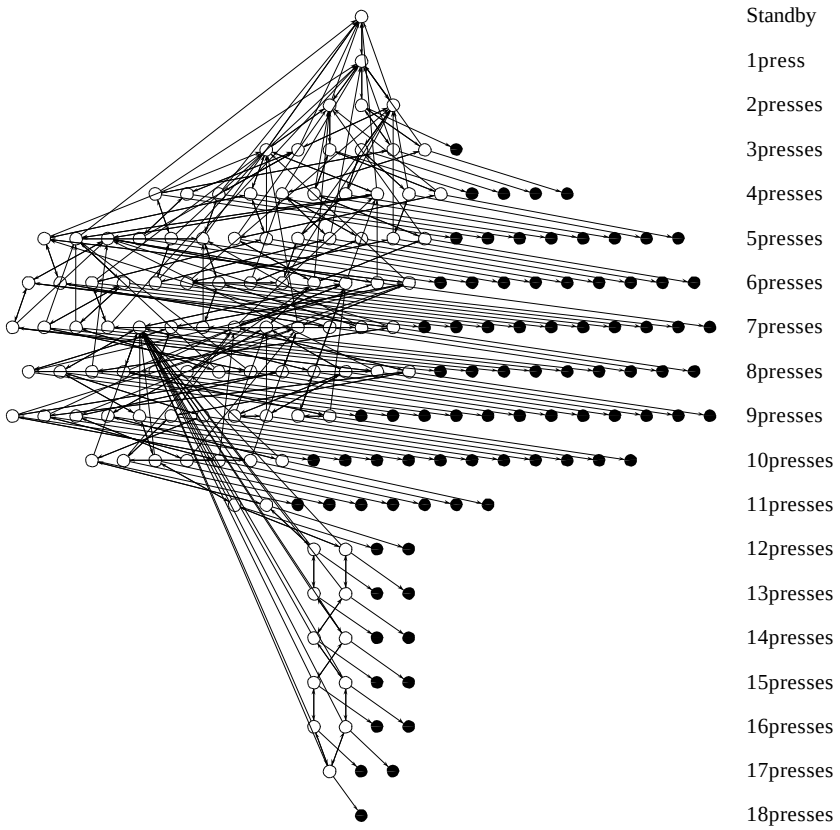

Figure 2. Visualising the cost of accessing Nokia menu functions.

Each arrow corresponds to a button press: pressing buttons takes the Nokia from one state to another. Since there are 188 states and four buttons, there are 752 arrows, but for clarity Figure 2 does not show arrows going from each of the 84 black circles back to *Standby* — otherwise the Figure gives an accurate idea of the complexity of the user interface that is the subject of this paper.

3 User interface simulation

The specification of the Nokia handset can be used to animate a complete working simulation of the user interface. Interactively, a user can press buttons and the simulated display will show what the Nokia would have shown. Additionally, the simulation can be instrumented so that it collects statistics on user behaviour.

The *Mathematica* code required to run the user interface simulation is simple and brief. After a few lines of support code (see Appendix 4), the following two panels provide the interactive functionality of the user interface: clicking on the buttons makes it work. Both panels are in fact *Mathematica* code. The complete keypad is about 20 lines of code, including specifying the keytop fonts and sizes, plus the (largish) data structure — also *Mathematica* code — for the 300dpi graphics symbol on the 1 key.

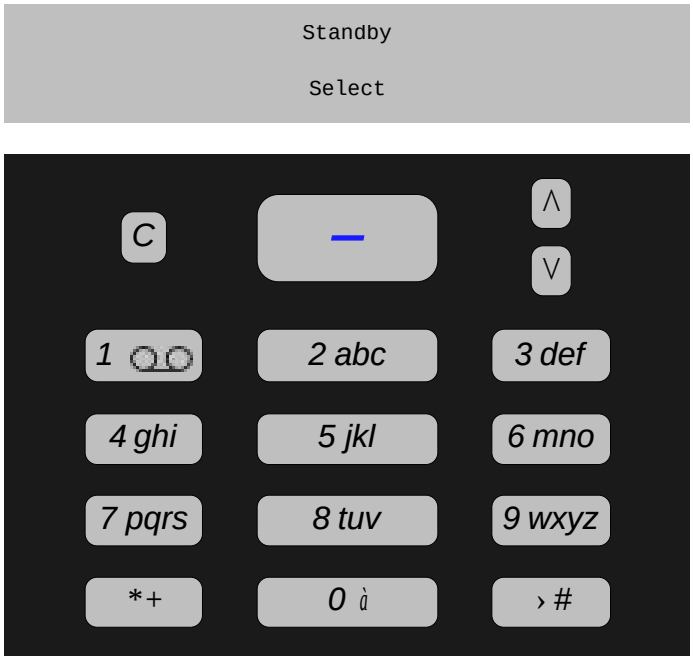

Figure 3. Interactive *Mathematica* simulation of a handset device.

4 User interface analysis

A single specification, for the Nokia 5110 menu functions, can be used to support a variety of analyses, as well as provide the basis for generating novel user interfaces that provide the same functionality.

Usability depends on many factors. The analyses, below, while not exhaustive of the sorts of mathematical questions that can be raised, are based on key press costs. A keystroke model could be used to estimate time, but this would take us beyond the space available for this paper; see Silfverberg *et al.*, 2000 (whose formula gives 240ms per keystroke, assuming skilled, continuous use of the index finger to press the menu keys). Another measure of usability is the probability that a particular key is used: the Nokia 5110 design appears to have attempted to increase the frequency the Navi key is used — it is a soft key, and reduces the number of other keys required. This creates the visual impression of a simple user interface as well as reducing finger movement. Yet it also means that menu functions (such as the phone's calculator) are inaccessible during phone calls, because in this mode the Navi key ends the phone call. Whether users need, say, a calculator during a phone call and whether this need should override the keypad aesthetics is an empirical question beyond the scope of mathematical analysis. Whatever we choose to analyse mathematically, design trade-offs can be formulated, which then raise interesting insights and questions that suggest further empirical work…

4.1 Goal weights

From the Nokia specification we can work out the optimal key press sequences to activate any function. The expected optimal number of presses is 8.83±3.29, meaning that if the Nokia is used optimally without error, then users will take 8.83 presses on average to activate menu functions with a standard deviation of 3.29. But of course, a real user will access some commands infrequently — especially the ones that Nokia have made less accessible. For example, it takes 11 presses to change the phone's security settings, as against the *Search* function, which only requires 3 presses to access. We would get a more realistic expectation of the number of presses if they were weighted by how likely each function is required by a user.

We could use the simulated handset to obtain weights by getting users to run simulated tasks, but this would take a long time (and many users), as well as begging the question where we could get appropriate distributions of tasks. No doubt

Nokia has, over time, collected enough statistics of use to do this accurately. If we had such figures, we could use them. Instead, for the purposes of this paper, it is sufficient to obtain a plausible probability distribution.

We will assume Nokia has arranged things so that more likely, more frequently used, functions take fewer key presses to activate. The Zipf distribution (Zipf, 1949) meets the requirements and is easy to calculate; moreover, the Zipf distribution occurs naturally in many contexts (e.g., it relates the length and frequency of English words) — the frequency of an item is inversely proportional to its cost. Weighting presses by the Zipf probabilities, we obtain an expected number of presses of 7.15±2.95. This number is of course less than the unweighted expectation because we have chosen a probability distribution that makes large numbers less likely.

Figure 4 shows an extract from the phone's functions, ranks, costs and Zipf probabilities, ordered by rank. (With all functions shown the probabilities would sum to 1.)

Functionname	Rank	Presses	Probability
Search	1	3	0.0613
Incomingcallalert	2	4	0.0306
Inbox	2	4	0.0306
Speeddials	2	4	0.0306
Servicenos	2	4	0.0306
....	....	....	....
Português	14	16	0.00438
Svenska	14	16	0.00438
Español	15	17	0.00408
Norsk	15	17	0.00408
Suomi	16	18	0.00383

Figure 4. Summary of functions, ranked by presses and Zipf probabilities.

Given the state probabilities, and other assumptions such as the probability of making errors and of pressing C, we could work out button probabilities. Without known error rates, for this paper we take the probabilities of pressing buttons to be equiprobable (i.e., 0.25).

The Nokia allows users to exit some functions returning to the previous position in the menu hierarchy, whereas others enter different modes or return to standby. For example, the search function can be used to look up a phone number, which is then dialled. At the end of the phone call, the phone is back in *Standby*, rather than returning to the phone book part of the menu hierarchy. We will assume, for uniformity, that when the user has accessed a goal state, with probability 1 on the next button press the device is returned to *Standby*.

4.2 Cost of knowledge graphs

There are many ways to analyse a user interface from its specification. The cost of knowledge graph was introduced and justified for usability analysis by Card *et al*. (1994) to visualise how easily a user can access the state space of a system. The graph shows the number of goal states a user can access against the number of user actions, that is, the cost of acquiring the knowledge available in each state. The cost of knowledge graph can be constructed from empirical data, from cognitive analysis, or analytically, as we now do. Our approach is probabilistic and does not assume error-free behaviour; the more realistic the probabilities used, the more realistic the evaluations that can be drawn from them. Details of the mathematics are given in Appendix 1 — for practical purposes (e.g., use by designers, rather than HCI researchers), what is important is the visualisation, rather than the way it is calculated; indeed, for anyone using this paper in its full *Mathematica* form, all that is necessary is to invoke a function that has already been defined.

A cost of knowledge graph for the Nokia menu system is shown in Figure 5. The solid line shows an unweighted cost of knowledge graph, but weighting (by the Zipf probabilities) gives a more realistic measure of knowledge — since the

user is less interested in some functions than others, and the Zipf probabilities reflect this well. The dashed line shows the weighted cost of knowledge graph.

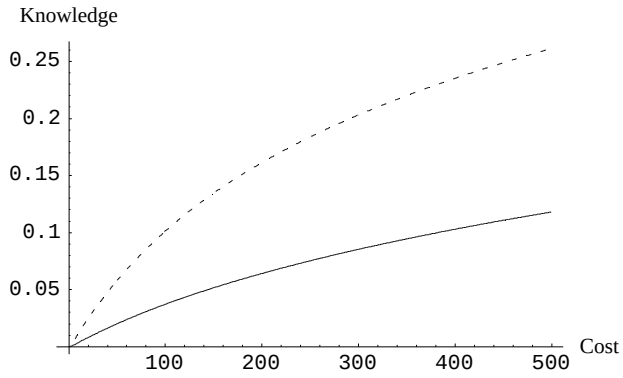

Figure 5. Cost of knowledge graph for the Nokia 5110 function menu. Dashed line is Zipf weights; solid line is uniform weights

The analysis could be refined. For example, we took the probability of pressing the C key as 0.25, which is possibly too high. Nevertheless, the point demonstrated is that with data (whether empirical or estimated) useful insights can be derived. Here we see, for instance, that in "average" use (i.e., as might occur in field studies) to achieve a coverage of 25% takes 455 button presses. (This figure does not translate linearly into a time, since the cost of knowledge assumes the user acquires knowledge, and thus pauses in each new state.)

Furnas (1997) suggests the pair (maximal outdegree, diameter) is a good indicator of the usability of a device; the original Nokia is (4, 19), compared to the Huffman tree alternative using the same keys discussed below, which is (4, 8). The digit key Huffman tree, also discussed below, is (11, 5) — showing that when more keys are used (here 10 digit keys, 0–9, and one correction key, C) the worst distance between states (the diameter, 5) can be considerably reduced. Other measures can be obtained from the specification. For example, to test whether every button works in every state takes a *minimum* of 3914 presses, assuming error-free performance. Such a high number suggests that human testing is inadequate.

4.3 Alternative user interfaces

A mobile phone can be controlled with many sorts of user interface. In this paper, following Nokia, we restrict ourselves to tree-structured interfaces. There are other alternatives, which can be much more effective — see: Marsden, Thimbleby, Jones and Gillary (2000), and Thimbleby (1997) for examples.

The Nokia uses four keys to select from 84 functions. If reducing the number of keystrokes was a design goal, then a Huffman tree (Huffman, 1952) is the most efficient way, in terms of keystrokes, to do this. From the original list of functions, we can construct a Huffman tree, using three keys (for navigation) and one key, retaining the C key, for correcting errors. Under these assumptions, we achieve an expected optimal number of presses of 4.04±0.53 (or 4.18±0.52 unweighted). See Figure 6 for comparison with the ranking of Nokia functions (Figure 4).

Functionname	Rank	Presses	Probability
Search	1	3	0.0243
Inbox	1	3	0.0243
Incomingcallalert	1	3	0.0243
Speeddials	1	3	0.0243
Servicenos	1	3	0.0243
....	....	....	....
Norsk	3	5	0.0081
Español	3	5	0.0081
Suomi	3	5	0.0081

Figure 6. Summary of Huffman costs and probabilities.

The entries in the Huffman table are in the same overall order as the original Nokia table; this is a consequence of building the Huffman interface on the Zipf probabilities, which were in turn inversely proportional to the ranks of the functions in the original design.

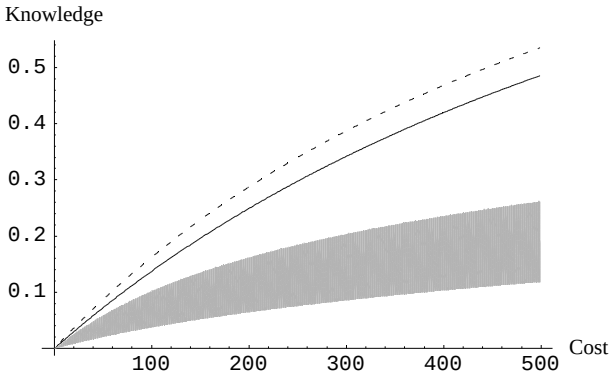

Figure 7. Cost of knowledge graph for the Huffman tree interface (the dashed line is Zipf weights; the solid line is uniform weights). For comparison, the lower grey region represents the range of data from the original Nokia as shown in Figure 5.

Figure 7 (note the different vertical scales compared to Figure 5) shows the cost of knowledge graph for the Huffman tree interface is *considerably* better (in terms of speed of access to the device's functions) than the original design; for example it achieves 25% coverage after 168 presses (compared to 455 presses for the original Nokia). Even so, the model over-estimates the costs because of the assumption of pressing the C key with probability 0.25.

A Huffman tree organises functions according to their probability, which would be convenient for a highly skilled user, but might seem arbitrary to a conventional user. There are two obvious improvements: first, the probabilities could be determined from the actual user's operation of the handset — the user interface would then adapt to be optimal for the user's own patterns of behaviour; secondly, the navigation keys could be left as they are (i.e., with a more-or-less topic-organised structure) and the numeric keys could be used for rapid access, providing shortcut codes.

In fact, the Nokia already uses numeric keys for faster access; some numeric codes are shown in Figure 1. Although the Nokia allocation of numeric codes corresponds to the menu structure, since the menu structure itself is of no interest to users, the codes are effectively arbitrary — for example, even if a user knows Navi 2 4 1, they are not likely to be able to work out what Navi 2 4 2 is! If we use ten numeric keys and the C key instead of just three navigation keys (i.e., creating a Huffman tree with fan-out of 10), we can access functions with 2.98±0.26 (3.01±0.24 unweighted) key presses. Note that using the digit keys means that a menu key is required (otherwise the digits pressed would just dial a phone number), compared to using the dedicated keys, where any of them can be pressed immediately without ambiguity. This adds 1 to the costs, which is included in the figures. In comparison, the original Nokia shortcuts have expected optimum presses of 3.39±0.92 (3.64±0.92 unweighted).

We can use *unallocated* fast codes from the Nokia design (for example, Navi 1 7 1 is already allocated, but Navi 1 6 1 is not allocated) and achieve an expected number of presses of 3.09±0.45 (3.29±0.48 unweighted) — which is marginally faster than Nokia's shortcuts. Since these codes are all different from Nokia's, we could have both schemes available at the same time (if we wanted to), so each function would have two codes, Nokia's original and the faster, unallocated codes. Since the codes are different, there is no confusion: either could be used. Since the user presumably doesn't care what the shortcut codes are, they could use Nokia's shortcuts if these are better, or the unallocated codes if these are better. This is having the best of both worlds, and unsurprisingly it works out even faster — at 2.69±0.46 (or 2.87±0.34 unweighted).

Device	Min	Max	Weighted	Unweighted
Nokianavigablemenu	3	18	7.15 ± 2.95	8.83 ± 3.29
Huffman, 3keys	3	5	4.04 ± 0.53	4.18 ± 0.52
Nokiadigitshortcuts	2	5	3.39 ± 0.92	3.64 ± 0.92
Unallocatedcodes	2	4	3.09 ± 0.45	3.29 ± 0.48
Huffman, 10digitkeys	2	4	2.98 ± 0.26	3.01 ± 0.24
Shortestcodes	2	3	2.69 ± 0.46	2.87 ± 0.34

Figure 8. Summary of expected optimal costs of accessing all goals.

With reference to Figure 8, there appear to be two errors. The shortest codes have a maximum length shorter than either the Nokia shortcuts or the Unallocated codes, yet it is based on both of them. The explanation is that the Unallocated codes do not take advantage of any of the Nokia's short codes, so some are quite long; the Shortest codes approach makes use of the short Nokia codes, and then has spare short codes of length 3 to replace the longer Nokia codes of length 4 and 5. The second apparent error is that the maximum length of the shortest codes is 3, but the maximum length of the Huffman codes is 4. Yet Huffman codes are theoretically shortest — so something must have gone wrong. The explanation is that Huffman codes are unambiguous (they are prefix free), but the Nokia itself is not. For example, if a user presses Navi 1 7 2 they get *Memory status*, but if they press Navi 1 7, they get *Options* (which then autocompletes to *Name number*, which has shortcut Navi 1 7 1 2, not Navi 1 7 1 1 as might be expected), even though this is a prefix of Navi 1 7 2. To avoid this apparent ambiguity, the Nokia effectively has an extra user action, DEL, when the user delays (if there is no delay, the user can use other keys; for example, Navi 1 7 ∨ is the same as Navi 1 7 2). Thus *Type of view* really has shortcut Navi 1 7 DEL, which is not a prefix of Navi 1 7 2. Thus the shortest codes cost is based on having, effectively, 12 keys (10 digits, DEL, plus C) as compared to the Huffman tree that makes do with 11 keys (10 digits plus C). Since the shortest codes are making use of more keys, the maximum length of a shortcut can legitimately be less than the maximum length of the Huffman code.

The shortest codes scheme has advantages — it preserves Nokia's original structure for the menu shortcut codes, and permits faster access where possible — and is better than the Hamming code approach, which has no advantages other than reducing key press counts. It is a design worth considering further and evaluating empirically. Figure 9 shows what the new codes look like, compared with the original Nokia codes (compare with Figure 1). For example, to access *Message validity*, the user can press either Navi 2 4 1 3, as specified by Nokia, or they can press Navi 2 7, saving two presses.The alternative menu codes in Figure 9 have been allocated so that shorter codes are allocated to higher Zipf probability functions.

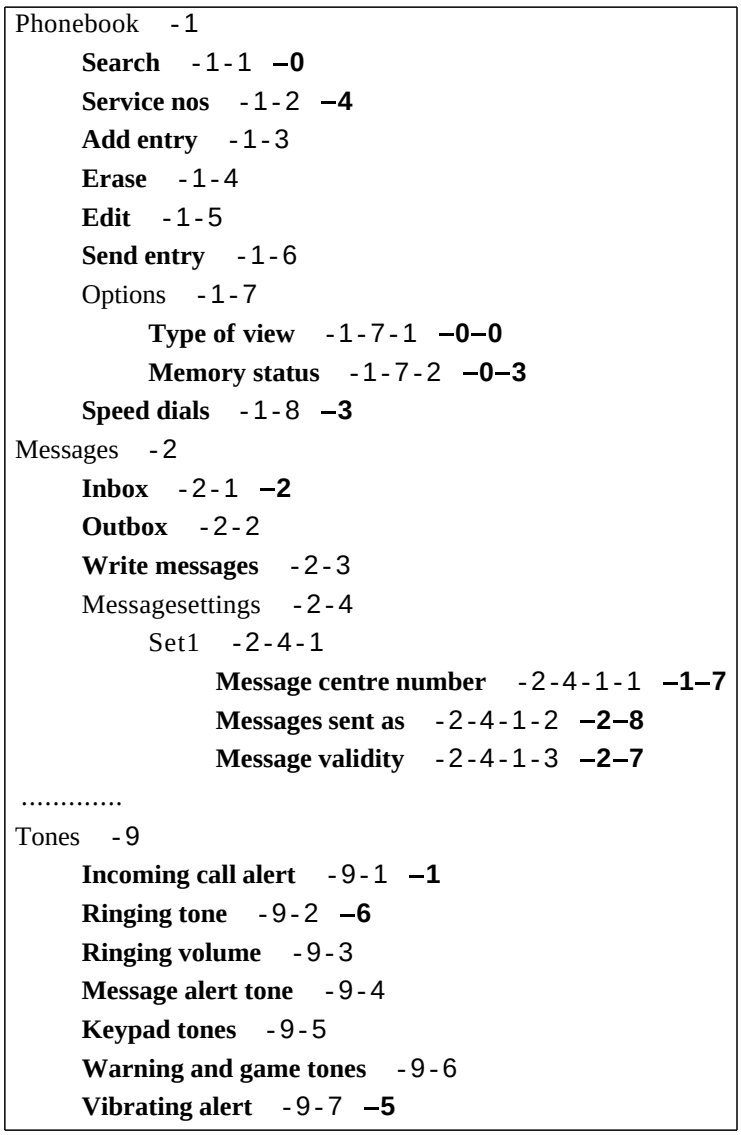

Figure 9. Extract from "shortest codes" menu. Both shortcuts can be used; shorter ones are in bold.

5 Conclusions

Despite being around for many years, and having had many opportunities for improvement, consumer electronics, such as video recorders and fax machines, are notorious for having poor user interfaces. Certainly it takes skill to perform usability evaluations well, and unfortunately the time pressures of manufacturing often mean usability considerations come too late to have any significant impact. Even if usability studies are done, in the time available, they are unable to cover entire designs — instead, practical evaluation concentrates on usability disasters or marketing features. Conventional usability engineering is relatively ineffective, whether because it is not used, or because it is used but is ineffective. The sorts of evaluations presented in this paper can be done automatically, by tools that are in any case required to specify and build working products. The paper showed that evaluations done in this way can raise and help explore interesting design issues.

More generally, the paper showed that a user interface can be specified and analysed in a paper. The work reported here is replicatable, and can be checked and developed easily. As it happened, the paper exploited *Mathematica*; very worthwhile further work would be to embed the appropriate *Mathematica*-like features inside design tools.

Acknowledgements

Ann Blandford and Matt Jones both made very valuable comments. Navi is a trademark of Nokia Mobile Phones.

References

S. K. Card, P. Pirolli and J. D. Mackinlay, "The Cost-of Knowledge Characteristic Function: Display Evaluation for Direct-Walk Dynamic Information Visualizations," *Proceedings of CHI'94*, pp238–244, 1994.

A. J. Dix, J. E. Finlay, G. D. Abowd and R. Beale, *Human-Computer Interaction*, 2nd. ed., Prentice Hall, 1998.

G. W. Furnas, "Effective View Navigation," *Proceedings ACM CHI'97*, pp367–374, 1997.

D. A. Huffman, "A Method for the Construction of Minimum-redundancy Codes," *Proceedings of the IRE*, **40**(9), pp1098-1952, 1952.

G. Marsden, H. W. Thimbleby, M. Jones and P. Gillary, "Successful User Interface Design from Efficient Computer Algorithms," *ACM CHI'2000, Extended Abstracts*, pp181–182, 2000.

M. Silfverberg, I. S. MacKenzie and P. Korhonen, "Predicting Text Entry Speed on Mobile Phones," *Proceedings ACM CHI*, pp9–16, 2000.

H. W. Thimbleby, "Combining Systems and Manuals," *Proceedings BCS Conference on Human-Computer Interaction*, **VIII**, HCI'93, pp479–488, 1993.

H. W. Thimbleby, "Formulating Usability," *ACM SIGCHI Bulletin*, **26**(2), pp59–64, 1994.

H. W. Thimbleby, "Design for a Fax," *Personal Technologies*, **1**(2), pp101–117, 1997.

H. W. Thimbleby, "Specification-Led Design," *Personal Technologies*, **4**(2), pp241–254, 1999.

K. Väänänen-Vainio-Mattila and S. Ruuska, "Designing Mobile Phones and Communicators for Consumers' Needs at Nokia," in *Information Appliances and Beyond: Interaction Design for Consumer Products*, ed. E. Bergman, pp169–204, Morgan-Kaufmann, 2000.

S. Wolfram, *The Mathematica Book*, 3rd. ed., Addison-Wesley, 1996.

G. K. Zipf, *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, 1949.

Appendix 1: The cost of knowledge graph

We are concerned with probability distributions of state occupancy. A vector v represents the probability of the device being in each state; we derived one such vector above, using Zipf probabilities. With a stochastic transition matrix P, if the distribution is v, one button press later it is vP. (A stochastic transition matrix is a transition matrix, where each transition is a probability. Each row sums to 1; the sum of the leading diagonal represents the probability of the device doing nothing in each state.)

If v_0 is the distribution at press zero (typically 1 in standby and zero in all other states) then $v_n = v_0 P^n$ is the distribution at press n. The probability of being in any state at any press, Pr(in state i at press p) = $v_p(i)$. The probability the device is not in state i at press p is 1–Pr(in state i at press p). Thus the probability it was never in state i over presses 0 to t is the product of these probabilities with p ranging over 0 to t. The probability is was sometime in state i is therefore 1 minus that:

$$1 - \prod_{p=0}^{t} (1 - \Pr(\text{in state } i \text{ at press } p))$$

The expected number of states visited to press t is the sum of these probabilities considered over all states. Since we are, more specifically, interested in the proportion of states visited to press t, the following formula is used for plotting the cost of knowledge function:

$$\text{knowledge}(t) = \sum_{i \in \text{States}} w_i \left(1 - \prod_{p=0}^{t} (1 - \Pr(\text{in state } i \text{ at press } p)) \right)$$

In this paper we take the state weights w to be the Zipf probabilities, or for 'unweighted' graphs weights from {0, 1/|Goals|} depending on whether the state is a goal state. Since the weights sum to 1, the measure of knowledge ranges over 0 to 1.

Appendix 2: Utility functions

This Appendix forms the common *Mathematica* code that creto create all the data structures (e.g., the transition matrices) from the basic definitions, which are given in Appendix 3. (It has to be placed before the definitions of the various devices.) To save space for the printed paper, all the code has been hidden.

Appendix 3: Device specifications

Nokia 5110

It is easiest to specify the Nokia 5110 by writing out a definition that is as close a match to the Nokia's user manual as possible. We then write a *Mathematica* function to convert this 'human readable' specification into a complete device specification.

```
readable[nokia] ^=
 menu["standby", {menu["phone book", {"search", "service nos", "add entry", "erase", "edit",
     "send entry", menu["options", {"type of view", "memory status"}], "speed dials"}],
   menu["messages", {"inbox", "outbox", "write messages", menu["message settings",
      {menu["set 1", {"message centre number", "messages sent as", "message validity"}],
       menu["common", {"delivery reports", "reply via same centre"}]}],
     "info service", "voice mailbox number"}], menu["call register",
    {"missed calls", "received calls", "dialled numbers", "erase recent calls",
     menu["show call duration", {"last call duration", "all calls' duration",
       "received calls' duration", "dialled calls' duration", "clear timers"}],
     menu["show call costs", {"last call cost", "all calls' cost", "clear counters"}],
     menu["call costs settings", {"call costs' limit", "show costs in"}]}], menu["settings",
    {menu["call settings", {"automatic redial", "speed dialling", "call waiting options",
       "own number sending", "automatic answer"}], menu["phone settings",
      {menu["language", {"Automatic", "Engish", "Deutsch", "Français", "Nederlands",
         "Italiano", "Dansk", "Svenska", "Norsk", "Suomi", "Español", "Português",
         "<Russian>", "Eesti", "Latviesu", "Lietuviu", "<Arabic>", "<Hebrew>"}],
       "cell info display", "welcome note", "network selection", "lights"}],
     menu["security settings", {"PIN code request", "fixed dialling", "closed user group",
       "phone security", "change access codes"}], "restore factory settings"}],
   menu["call divert", {"divert all calls without ringing", "divert when busy",
     "divert when not answered",
     "divert when phone off or no coverage", "cancel all diverts"}],
   menu["games", {"memory", "snake", "logic"}], "calculator",
   menu["clock", {"alarm clock", "clock settings"}],
   menu["tones", {"incoming call alert", "ringing tone", "ringing volume",
     "message alert tone", "keypad tones", "warning and game tones", "vibrating alert"}]}];
```

This readable tree structure is converted into a list of transitions using the following code. In this example, the button probabilities are set equal at 0.25, but other values can easily be used. The code also gives a name to the specification, as was used, for instance, in Figure 8.

```
convert[nokia] :=
 Module[{auxconvert, p = {}, goals = {}, transition}, transition[from_, button_, prob_, to_] :=
   AppendTo[p, {menuname[from], button, prob, menuname[to]}];
  auxconvert[menu[name_, items_]] := Module[{i}, transition[name, "Navi", 0.25, items[[1]]];
    For[i = 1, i £ Length@items, i++, transition[items[[i]], "Down", 0.25,
      items[[If[i == Length@items, 1, i + 1]]]; transition[items[[i]], "Up", 0.25,
      items[[If[i == 1, Length@items, i - 1]]]; transition[items[[i]], "C", 0.25, name];
     If[Head@items[[i]] === menu, auxconvert[items[[i]],
      AppendTo[goals, accessed@menuname[items[[i]]]];
      transition[items[[i]], "Navi", 0.25, Last@goals];
      Map[transition[Last@goals, #, 0.25, "standby"] &, {"C", "Navi", "Up", "Down"}]
     ]
    ]
   ];
  auxconvert[readable[nokia]];
  symbolicGoals[nokia] ^= goals;
  startState[nokia] ^= "standby";
  Map[transition["standby", #, 0.25, "standby"] &, {"C", "Up", "Down"}];
  symbolicTransitions[nokia] ^= p;
  initialise[nokia, "Nokia navigable menu"];
 ];
convert[nokia];
```

The final line of code converts the symbolic specification into all the forms required by the body of the paper. It also performs various internal checks (e.g., that probabilities add to 1).

Other device specifications

The Huffman tree, the Nokia shortcuts and the other device specifications are built automatically from the Nokia specification (as defined in the previous section). The definitions are omitted in the printed version of this paper.

Appendix 4: User interface code

The user interface code is simple enough to be given in its *entirety*, even in the printed version of the paper. A function is defined to operate the LCD display panel, which itself is just a *Mathematica* paragraph defined with a grey background and black text, to simulate the Nokia's LCD appearance.

```
lcd[stateno_] := Module[{nb = InputNotebook[]},
  NotebookFind[ nb, "LCD", All, CellTags]; SelectionMove[nb, All, CellContents]; NotebookWrite[
   nb, GridBox[{{StyleBox[capitalise[FromStateNo[nokia, stateno] /. accessed[s_] ¶ "Do " <> s]]},
     {StyleBox["                    "]},
     {StyleBox[If[stateno ä startState[nokia], "Menu", "Select"]]}}]]
 ];
```

There is some special-case code to say when the user activates a function, to capitalise the first letter of state names, and to display the Navi button's prompt as "Menu" or "Select" depending on whether the handset is in the start state, *Standby*.

The `press` function, executed when the user presses any button, relies on a function `nextState` that takes the device (whatever it is) from one state to the next, depending on which button is pressed.

```
press[key_] := nextState[stateNumber, key]
```

The rules for the `nextState` function are generated from the handset's specification.

```
numericTransitions[nokia] /.
 {from_, button_, prob_, to_} ¶ (nextState[from, button] := lcd[stateNumber = to]);
```

It would be trivial to modify `press` so that button presses (and timings if required) were recorded for analysis. Finally it is necessary to initialise the state to the start state, and display the appropriate text for that state in the LCD panel:

```
lcd[stateNumber = ToStateNo[nokia, startState[nokia]]]
```