Clicks and pauses when encoding

Joris Verrips[1]

An experiment is described with the author as his own test subject to decide if, when encoding with a single switch, pauses before longer codes are longer. This is not true for every encoding and therefore the assumption is false. A click-level model of time spent is described and several other models related to encoding with either one or two switches.

Introduction

Text based speaking communication aids can be accessed by switches either using a form of scanning or of encoding. The latter has longer learning curves but is faster almost from the start and combines well with stored text. Encoding is the fastest input method with switches but is slower than eye gaze, feasible with a minority of those who need communiation aids. It is used, though rarely, in Augmentative and Alternative Communication, or AAC, combined with synthetic speech.

How to easily input data with a single switch way has been named the one key challenge (MacKenzie, 2010). Encoding does not really solve it because encoding has to be learned. Other techniques like groupwise scanning require visual attention and a display or time consuming auditory menus. One may wonder what encoding is most easily learned, how fast it is in a few hours practice and if we can understand encoding with models. We will concentrate on Alternative Code, explained in the next section, and will hardly consider individual differences in the rather varied user groups. Also, we will try to be brief.

Alternative Code

Figure 1 shows Alternative Code, or AC[i], has structure by design. Frequent codes are short and most consecutive characters are assigned to related codes. The code for n is … or three dits, and must be shorter than the code for c, that has three dashes. Figure 2 shows adapted Classical Morse Code. __ means a long hold or a sustained dash. It takes a bit more time than a dash and may be held. 1 = _ 2 = __ or ._ or _. 3 = ___, __., . __ or _.. and so forth.

```
SP  .     BS  -      , -.-.

 a  .-     b  ..--   c  ---    d  --     e  ..
 f  .--    g  -.-    h  --.    i  -.     j  -...
 k  .-..   l  ..-.   m  ...-   n  ...    o  ....
 p  .-.-   q  ---.   r  -..    s  .-.    t  ..-
 u  --..   v  -..-   w  .--.   x  .---   y  -.--
 z  --.-   Ret ----

 1  -....  2  .-...  3  ..-..  4  ...-.  5 ....-
 6  ..... BS* -__   3&  .._
```

Figure 1. Alternative Code or AC. * means repeatedly and & means etcetera.

---

1 Is a medical doctor with a background in mathematics and in computer science, who studies text based communication aids intended for neurological patients with progressive diseases. Mail j.verrips@planet.nl and site www.depratendecomputer.nl.

```
SP  ..    BS  ----   ,  ---.

 a  .-     b  -...   c  -.-.   d  -..    e  .
 f  ..-.   g  --.    h  ....   i  --     j  .---
 k  -.-    l  .-..   m  ..--   n  -.     o  ---
 p  .--.   q  --.-   r  .-.    s  ...    t  -
 u  ..--   v  ...-   w  .--    x  -..-   y  -.--
 z  -..    Ret .-.-

 1  .----  2  ..---  3  ...--  4  ....-  5 .....
 6  -....  7  --...  8  ---..  9  ----.  0 -----
```

Figure 2. Adapted Classical Morse Code or CMC.

Literature

Kieras, 2005, makes a case for so called intermediate level models of what users go through to inform redesign and evaluation of interactive software and other user interfaces. Such models are intended to inform re-design, not do psychology but to kind of apply it. Usually individual differences are ignored and they need not solve our question about learning, because they mostly model execution.

Miller, 1956, describes learning to encode by a telegraphist and King, 1999, is worth reading as well. Once mastered encoding becomes almost subconscious, fast, and automates on the word level or above. Output buffers may be observed as with other transcription tasks. When interrupted one typically has to first key in a few words before one may shift ones' attention. How fast encodings are learned is hard to say, some people achieve twenty chars per minute almost from the start when writing codes down. Even they need many hours before it can be effected with eyes closed and many more before encoding no longer requires conscious effort.

Code length

With code length is traditionally meant result of a formula. Every dit counts for one, every dash counts for three, one for every pause between clicks that may be left out, two for end of character and three for Space or end of word. This formula then estimates the time needed to execute a code with a single switch. As dashes take at least three times as much time as clicks, code lengths are very different from one character to another. The reader may verify that with AC codelength so defined is higher with less frequent characters as q or x than with more frequent ones like j, k, l or m. Frequent characters have less dashes and this on purpose. When Vail, in 1836, defined Classical Morse Code he must have used a related formula combined with frequencies of characters in printers lay for English. We may omit pauses between clicks without much consequence.

When discussing logfiles we will mean the time needed to click a code and wait for the machine to understand it. That is, *including* the pause that signals end of code. And, of course, including the shorter pauses between individual clicks. When used with computers it is better to use characters for Sp(ace), BS (BackSpace), Return and numerals, Classical Morse Code needed some adaptations. This encoding is only one of 30! assignments of 30 different codes to 30 different characters.

Diverse codes can be compared to diverse scanning algorithms with frequency weighted averages leading to clicks and pauses per character. See Verrips, 2012 and Verrips, 2013, more background is in Verrips, 2015, Verrips 2016. This can also be done for the complexity of different encodings, assuming we have reasonable estimates of the complexity of individual codes. One may also do

experiments with a copy task, logging clicks and pauses.

Exercise with logfiles and a copy task

Recently some improvements to AC were found, making it more consistent than before. Its design now seems finished and was based in part on reading of literature on HCI but also on modelling in evaluation of user interfaces. Logging click times and pauses using several forms of scanning and of encoding with a single switch were also helpful. Recently, pauses *before* a code was entered seemed *longer* when the code was also *longer*. This suggests that longer codes are harder to remember, related to something that I had read somewhere, long ago. Code length and pauses can be logged and stored for subsequent presentation. Figure 3 and Figure 4 show data after an informal copy task of poetry with a single switch and Alternative Code.

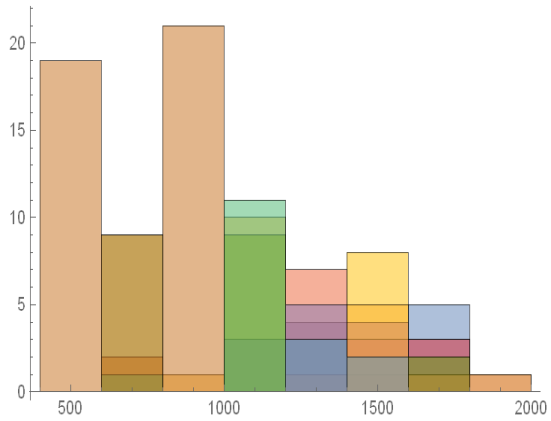

Figure 3. Histogram of measured code lengths per character. Outliers over 2000 milliseconds were left out and pause time was 280 milliseconds. Different characters contribute to different coloured blocks.

Figure 3 offers code lengths per character measured in milliseconds and including pause time after each code. Values are wide apart, as one would expect from Figure 1 once it is realised that dashes take much more time than dits, say 350 milliseconds instead of 110 milliseconds.

Pause length

P1 to P4, with p from previous, are pauses that came *just before* codes of length 1 to 4 were clicked in. SDs are large and it appears that longer codes are preceded by longer pauses in milliseconds.

P1 Mean 524.49 SD 264.21	P2 Mean 670.98 SD 589.12
P3 Mean 743.34 SD 557.85	P4 Mean 1036.15 SD 900.55

Figure 4. Data from informal measurement copying text that made us wonder.

We see means P1 < P2 < P3 < P4 with high SD's and assumed this must be more pronounced in early learning because short codes are easier, more frequent, and less of them too. So we felt tempted to investigate if this is always true. If .... is the code for a spoken o and we select it twice with pause time pt we have P4 …. pt o P4 …. pt o. Figure 5 offers more detail with clicks shown as dots . . in time shown as - - -

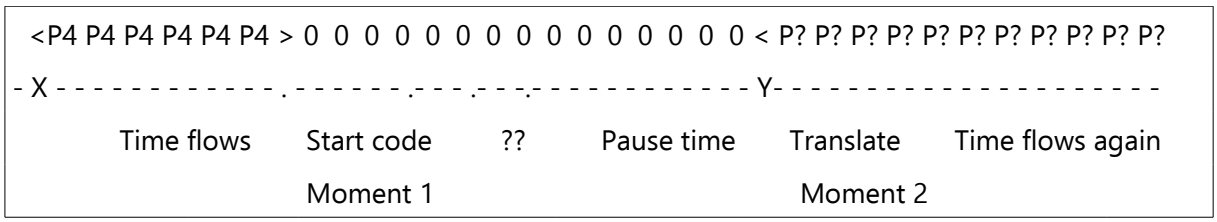

Figure 5. Events ordered in time, shown from left to right. ?? is explained below.

In Figure 5, code length is marked as 0 0 0 0. Time - - - flows from left to right with clicks . and Y is the moment that the machine translates the code into a character. We will assume that the previous code was translated at X, the next one will be translated on the far right of the page. We do not know its length, therefore note P? instead of P1 or P2 or P3 or P4. A code that consists of four dits or .... was translated by the machine at Y because no more dits or dahs were entered for a while. This pause time can of course be adapted[ii]. If we measure the time between Moment 2 and Moment 1 and subtract the pause time, we have the code length.

Pauses must vary and this may depend on the length of codes that must be retrieved from memory. Novices often start with a code and do not memorize it, not even read all of the code. So they need two lookups and are amazed the machine starts translating half of a code during the second one. In Figure 5, if they look at the screen from moment ?? onwards, that is before they enter the fourth dit, after a while the machine may translate the code ... into the character n instead of …. into the character o.

We model that X to Y is the time a user needs to encode the character o. This is doubtful with a transcription task due to psychological output buffers. Therefore neither novices nor experts are modelled with precision in Figure 6 just below.

```
Read or remember next word

 For each character from left to right
    Lookup the code or Remember the code
                        // Now follows a pause P?
                        // Meaning P1 or P2 or P3 or P4
                        //  and before the actual encoding
    Click it in         // Pause between codes is now over
                        // clicking a code will take about a second
    Verify character (optional)
    Forget code
 Remember and enter code for space

 Verify word (optional)

 Closure / Forget / Relax / Next word
```

Figure 6. Simplified sub-keystroke level model. Errors and learning were left out.

Closure is used in several meanings in the literature. In Card, Moran and Newell, 1983 a closure may occur when a part of a task is finished. This can often be observed by introspection. When learning one needs some time to observe and possibly turn ones' attention elsewhere. At table tennis, after every few turns some people may be heard to sigh. However, not all people, and when exactly the closure ends is hard to say. Intuitively, closures are related to learning and the need for

closure diminishes when one learns something. Instead of serial processes, in some fields of study people model concurrent processes, the example below is in Baker and Hengeveld, 2013. One may hum a song, think of something, cycle *and* look around. This casts doubt on the strictly serial model in Figure 6.

Thinking > Thinking > Thinking . . . . .

informs

Formulating > Formulating > Formulating . . . .

informs

Speaking > Speaking > Speaking . . . .

Figure 7. Other models exist that are *less* appropriate to model input but *more* to analyse speaking.

Figure 8 shows histograms of P1 to P4 without values above 2000 milliseconds. Pauses are at least 280 milliseconds, the pause time. Because histograms rise steeply on the left side higher speed is expected with a lower pause time.

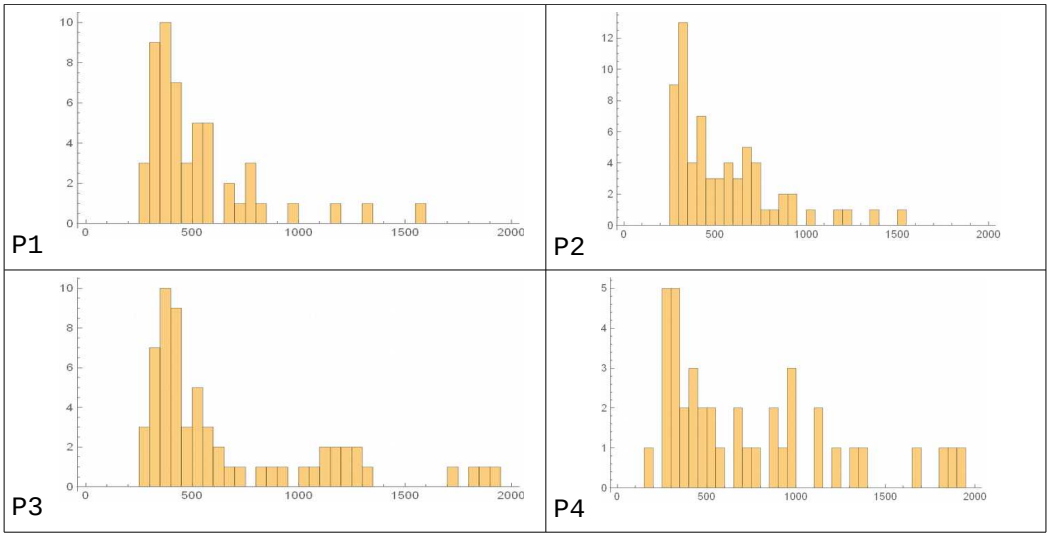

Figure 8. Histograms of pauses *before* codes of lengths 1 to 4 or P1 to P4 and with AC.

Hypothesis

It sounds reasonable that to remember a longer code, and distinguish it from more codes of that length, will take more time. Rare codes also may tempt one to do a Lookup, a time consuming process. So we have several reasons for P4 > P1, our hypothesis.

P1 < P4 might well be more pronounced with encodings that lack structure (like adapted classical Morse Code or CMC) especially while learning it. So we may verify the hypothesis with CMC.

Criticism

That long codes have longer pauses previous to them need not be true all of the time or in every phase of learning for every individual and for every code. One might assume that the pause time allows a partial look ahead before short codes. The effect might disappear or express itself rather differently with a shorter pause time. Also, it has no immediate consequence for re-design. But we were curious enough to do a small experiment and report on it below.

Method

I entered a Dutch poem[iii] I already knew with AC, without noticing an output buffer and concluded that my learning of AC is far from finished. I then trained a few hours with CMC, that I had not recently used, and quickly picked it up again[iv] Once I could enter the alphabet forwards then backwards with eyes closed I also entered the poem twice, the first time with many errors the second time with two words repeatedly entered and acceptable number of errors. In both cases, all errors were corrected. Using oneself as a test subject implies researchers bias[v].

Results of experiment

Figure 9 to Figure 13 show data extracted from logfiles of the experiments.

AC with single switch	CMC with single switch after partial unlearning of AC
28 Backspaces, 96 Spaces, 515 characters 1358 Clicks, 939 dits and 419 dashes 996 Total hands-on time in seconds Characters per minute: 31.024 Clicks per character: 2.64 Pauses: 140/280 milliseconds.	28 Backspaces, 94 Spaces, 535 characters 1314 Clicks, 881 dits and 533 dashes 1260 Total hands-on time in seconds Characters per minute: 24.095 Clicks per character: 2.60 Pauses: 140/280 milliseconds.

Figure 9. Logfiles overview.

After holding for 140 milliseconds a dit changes in a dash and 280 milliseconds after releasing the code is interpreted by the software. The number of clicks per character with AC, 2.6, agrees with modelling based on relative character frequencies, see Verrips 2013. That in both columns we have 28 BackSpaces is a coincidence.

P1 No 124 Mean 422.16 SD 285.75	P2 No 171 Mean 511.34 SD 383.64
P3 No 138 Mean 634.77 SD 583.73	P4 No 114 Mean 937.33 SD 777.97

Figure 10. P1 to P4 in milliseconds with AC.

In Figure 10, 124 codes of length one were given, Spaces and BackSpaces, codes . and – and only 114 codes of length 4, there are sixteen of them. We have, and did expect, P1 < P2 < P3 < P4, but these differences are not significant. SDs are high.

Q1 Mean 1018.75 SD 748.14	Q2 Mean 577.93 SD 555.13
Q3 Mean 374.85 SD 188.13	Q4 Mean 524.28 SD 368.62

Figure 11. To verify pauses were also registered with the lengths of the previous codes. This we call Q1 to Q4. We do *not* have, and did *not* expect, Q1 < Q2 < Q3 < Q4.

P1 100 Mean 938.55 SD 629.20	P2 210 Mean 938.27 SD 675.19
P3 136 Mean 1177.44 SD 784.77	P4 79 Mean 1031.25 SD 632.35

Figure 12. P1 to P4 with CMC.

Chars	Avg	SD	Chars	Avg	SD
a	1041.88,	49.94	m	1564.31,	70.31
b	1715.67,	32.37	n	1122.43,	36.06
Bksp	771.46,	37.25	o	1438.71,	50.65
c	1700.33,	28.30	p	1804.33,	47.71
d	1196.50,	40.68	r	1308.52,	44.06
e	854.50,	25.28	s	1320.77,	49.02
f	1443.01,	39.59	Sp	586.35,	22.23
g	1486.45,	66.13	t	1269,00	32.71
h	1502.18,	62.00	u	1691.43,	49.20
i	1048.21,	38.45	v	1771.14,	58.55
j	1590.50,	44.84	w	1743.13,	59.13
k	1572.23,	77.37	z	1896.33,	88.16
l	1581.11,	79.42			

Figure 13. Measured code lengths, including pause time, averages and SD per character. This is from the experiment with copy task and AC. Characters q, x and y were not entered.

Figure 14 and Figure 15 illustrate something is wrong with our hypothesis.

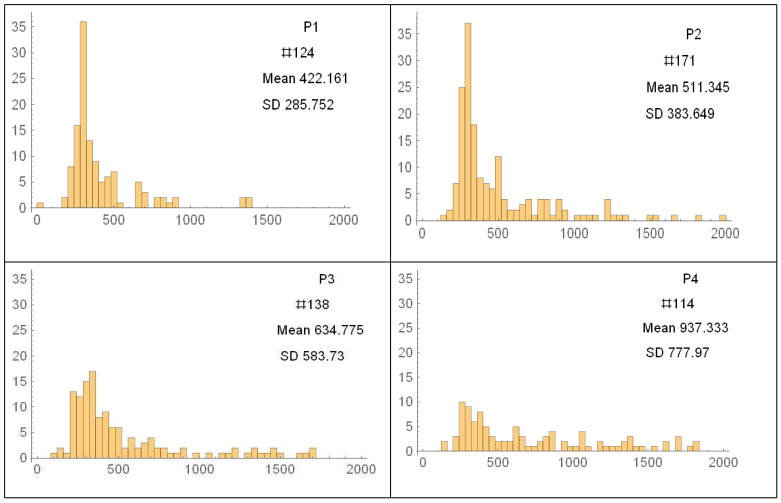

Figure 14. Histograms of P1 to P4 with AC and data from Figure 10.

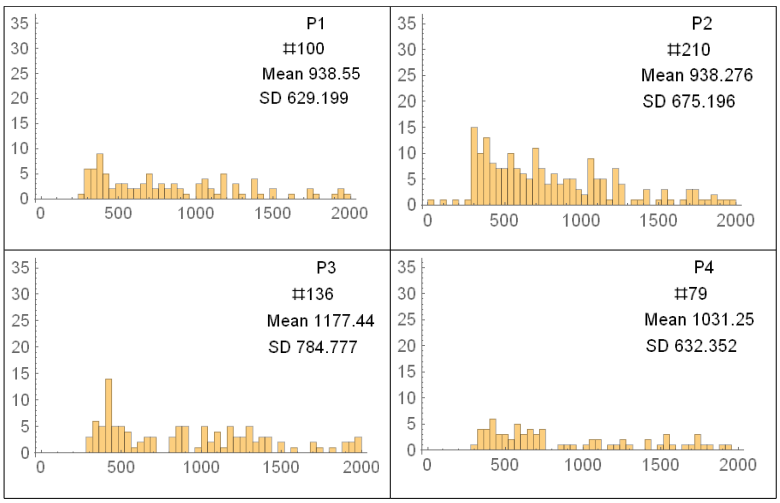

Figure 15. Histograms of P1 to P4 with CMC and data from Figure 12.

Interpretation

Code lengths in Figure 13 have small Standard Deviations with an average of 3.5%. As we felt like we were encoding a character at a time and used many BackSpaces, learning has not ended. We do *not* see P1 < P2 < P3 < P4 with CMC. And we did *not* do a reliable comparison between AC and CMC[vi] because we trained more with AC. Input is a so called 'wicked problem' and we needed much practice to just vaguely understand encoding. That our hypothesis was wrong and that our psychological insights lack precision should therefore not amaze us much. Both Moran and Card, 1982 and Moran and Carroll, 1996 offer more insight on these issues.

Click level model

We may do a click level model, that is *below* the keystroke level model as described by Card, Moran and Newell, 1983. This should help us to predict execution time based on the number of clicks needed. We will be inspired by numbers in Figure 9 and must multiply or scale by about 2, because keystroke level models explain but part of the time spent in routine tasks. People do more than just execute tasks and we should be glad to explain half the time spent.

Assuming #dits means the number of dits measured, #dashes likewise, dits take 100 and dashes take 300 milliseconds, and pauses between clicks that make up codes are 100 milliseconds, we may estimate total time with a single switch and pause time of 280 milliseconds. The model looks simple: Time spent = (#dits *100 + # dahs * 300 + (#dits + #dashes - #characters) * 100 + #pauses * 280)* 2. With two switches we have likewise (#dits * 100 + #dashes * 100 + (#dits + #dashes - #characters) * 100 + #pauses * 280) * 2.

Theoretical considerations suggest that not all codes will have the same lookup time, see below, and Figure 6 suggests that the pause before the *first* character of a word may well be longer, due to spelling of the next one. Likewise, if we postulate a verify operator before macros are selected, we would expect that pauses before space have a bicuspid distribution and likewise for sustained dashes. On the click level such considerations are *not* taken into account.

Testing the click level model

That the model is rather reasonable is shown in Figure 16, to be compared with Figure 13, there seems no need for statistical tests. We scaled with 1.7 instead of 2 but still compute somewhat higher values as in Figure 13. This is understandable because in the experiment we used a rather long pause time of 280 milliseconds while the test subject is faster with shorter pause time. A long pause time would allow some thinking or look-ahead to occur in time that is modelled as an inevitable pause. We see a = .- = (dit + dash + pause) * 1.7 = (100+300+280) * 1.7 = 680 * 1.7 = 1156.

a 1156. .-	N 986. ...
b 1836. ..--	o 1156. ....
Bksp 986. -	p 1836. .--.
c 2006. ---	q 2176. ---.
d 1496. --	r 1326. -..
e 816. ..	s 1326. .-.
f 1666. .--	Sp 646. .
g 1666. -.-	t 1326. ..-
h 1666. –-.	u 1836. --..
i 1156. -.	v 1836. .--.
j 1496. -...	w 1836. -..-
k 1496. .-..	x 2176. .---
l 1496. ..-.	y 2176. -.--
m 1496. ...-	z 2176. --.-

Figure 16. Computed code lengths according to formulas in the literature and with Alternative Code that is also shown.

One switch or two?

We note that the difference between one and two switches, though easily noted, is in the order of 200 milliseconds * #dashes in the model above. With #dashes taken from Figure 9 this boils down to about ninety seconds in experiments that took sixteen or twenty one minutes to complete or only about 10%. By design, codes of frequent characters have fewer dashes than less frequent characters.

Experiments have more parameters than #dits and #dashes, important inter individual differences exist and copy tasks do not faithfully represent communicating with a speaking hi-tech aid. The average statistician will remark all such parameters are *stochastic* variables and that even P1 < P4 in Figure 4 is not statistically significant. Therefore we really need some psychology to model what happens. Modelling also adds uncertainty because assumptions on code length and on lookup time need not be true.

Numerals

From Figure 1 and Figure 2 we may learn that numerals 1 to 6 in CMC have more dashes than numerals from AC. The model learns us that the assignment in AC is somewhat faster. See Figure 17. In unpublished work, AC combined with sustained dashes appears a bit faster yet with the code for 1 (n=1).

Numerals	AC	Estimated click time in milliseconds	CMC	Estimated click time in milliseconds
1	-....	700	.----	1300
2	.-...	700	..---	1100
3	..-..	700	...--	900
4	...-.	700	….-	700
5	….-	700	…..	500
6	…..	500	-....	700
		Total 4000		Total 5200

Figure 17. Trying to apply the click level model. We assumed dit = 100, dash = 300 and pause = 0.

Conclusion

In a n = 1 study of clicks and pauses when encoding in a copy task, the author acted as his own test subject. We tried to verify that pauses before long codes would be longer. This appears not true. The effect was seen with an encoding mastered well but not to perfection. It was not seen at all with a different encoding mastered superficially. We falsified our hypothesis at low cost.

Post mortem

After the fact, the theory we used seems too simple. Therefore it seems not worth the effort to repeat the experiment with other test subjects, and at significant cost. Though of course, that *is* standard methodology. Also we gradually came to appreciate, again, that some codes are harder than others, suggesting work of a more theoretical nature that we might have supplemented with more literature study and with a few years of psychology-.

Verrips 2013 estimated code complexity based on the number of related codes. So h = …. in CMC would be related to BS = ---- and be less complex then l = .-.. related to -... to ..-. to ...- and even -.--- so there is more opportunity for confusion. This suggests that we should not have assumed that all codes of length 4 would all of them be harder to comprehend than all codes of length 3. It *might* be true once retrieving them from memory is automatic, due to training, but this moment is hard to point out because some will have been trained more than others. We will come back on this analysis in Figure 25 below.

A slower decrease of P4 than of P1 might show up in a long individual learning curve if the same task is done from time to time to measure input speed. We dare not bet that this allows to compare CMC and AC experimentally. Also, input mechanisms should be tested with the intended target groups that are extremely varied and can not comply to experimental protocol. One may however try to model.

Models of encoding

Kieras and Polson, 1985, state that simpler user interfaces must have simpler transition graphs. Figure 18 would be easier than Figure 19 because it is smaller. Although this sounds plausible it is not easily applied to a transcription task with rate enhancements. The graphs rapidly become

unsightly and difficult to reason about. Also, we need a psychological theory on encoding, not a general approach to user interfaces.

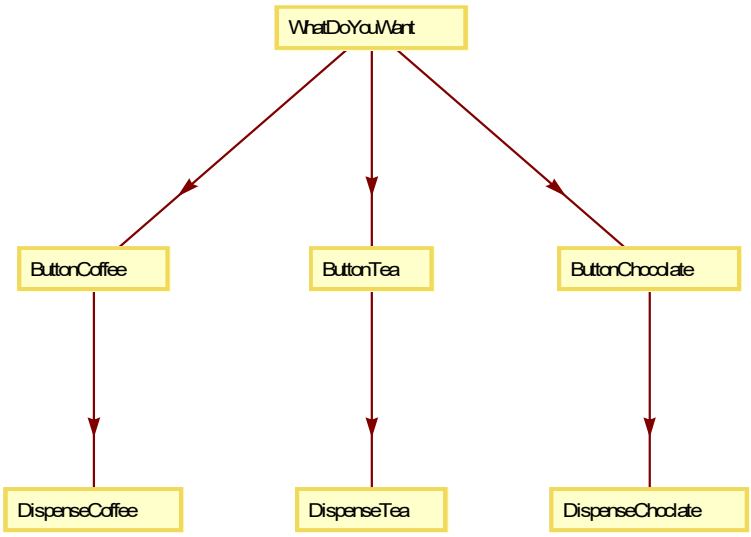

Figure 18. Coffee for free.

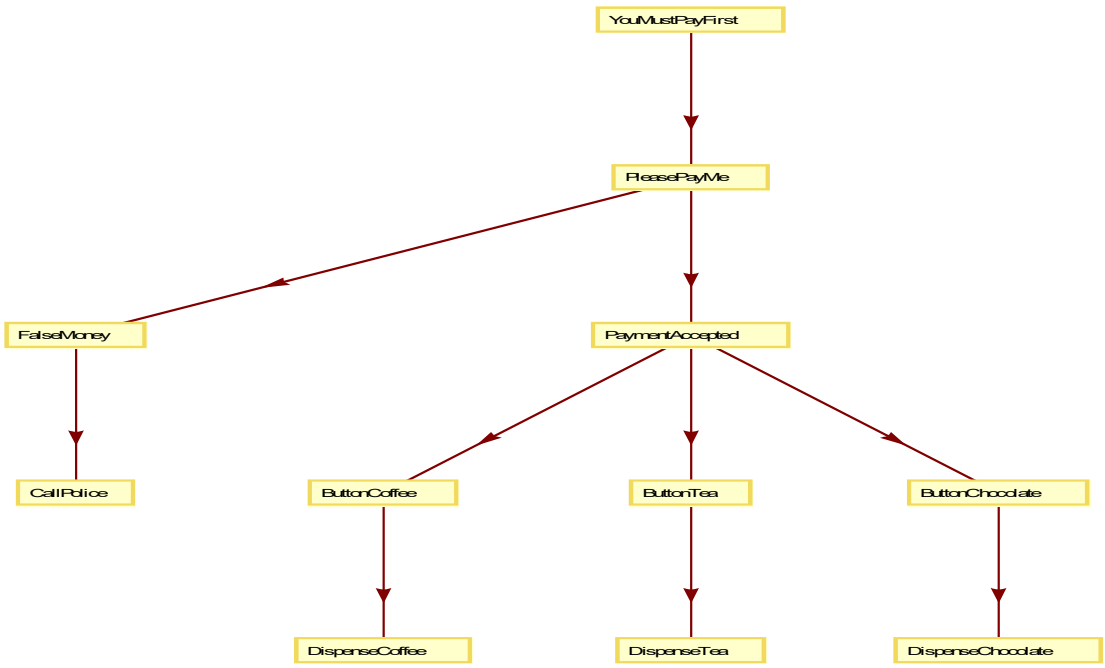

Figure 19. Coffee at a cost.

Hick's law comes to mind that states that choicetime is linear with the log of the number of alternatives with a small constant. As Log 28 = Log 4 + Log 7 Hick's law says that it does not matter much how we choose but that it does matter how many alternatives we have. In Figure 20 the second choices appear easier to make as most obey a simple rule. Possibly, consistent rules are learned more easily and the first one is easy. Little difference between AC and CMC is expected based on Hick's law.

As to Ebbinghaus, famous for studies on learning and forgetting, and who showed that nonsense

words are harder to learn, it is not clear if that applies here. First because fast retrieval is at stake second because it is not evident which codes would be more nonsensical, or complex. Or even if knowing one code might be helpful to guess another, or might help to rehearse it. Many teachers stress the importance of structure and meaning for learning.

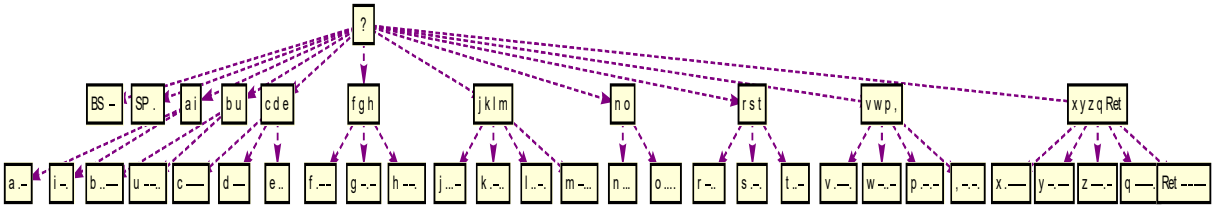

Figure 20. Two choice model for AC.

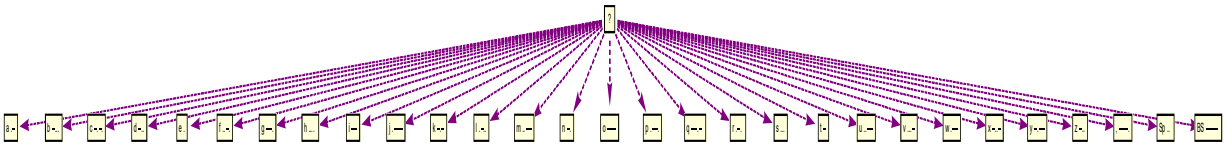

Figure 21. One choice model for CMC.

Experimental approach

In previous work much time was spent to make a consistent user experience and minimise *un*learning once new properties are activated by the user such as enriching word prediction with phrase prediction. We would have liked to document easy learning with able bodied test subjects and report something like Figure 22, displaying time spent practising horizontally, input rate vertically, and rapidly learned rate enhancements modelled as hyperbolas of the form v(t) = (a t + b) / (c t + d) + e with grossly the same effect on input rate. Intuitively hyperbolas are better models of learning complex tasks than exponential curves[vii].

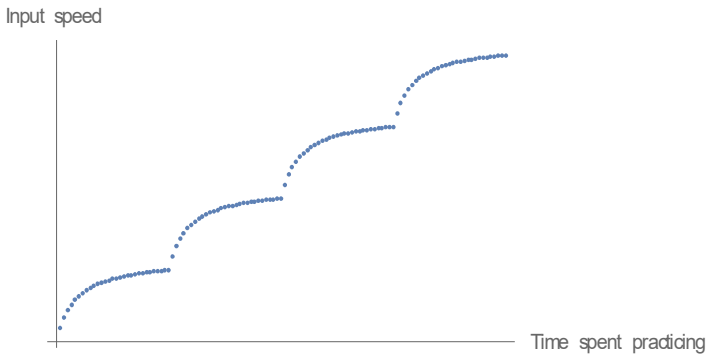

Figure 22. Hypothetical learning curve computed with a few lines of software[viii].

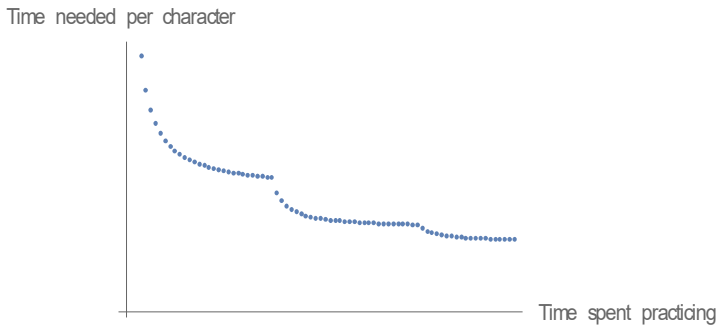

Figure 23. Hypothetical time per character due to learning. The asymptote represents codelength and overhead.

The dimensionality of the vertical axis in Figure 22 is character per time and in Figure 23 its inverse, time per character. Such graphs were far beyond us in Verrips 2013b, see Figure 24 with vertically the measured input rate with a single switch, WriteEasy and rate enhancements in slow open conversations with two speaking computers.

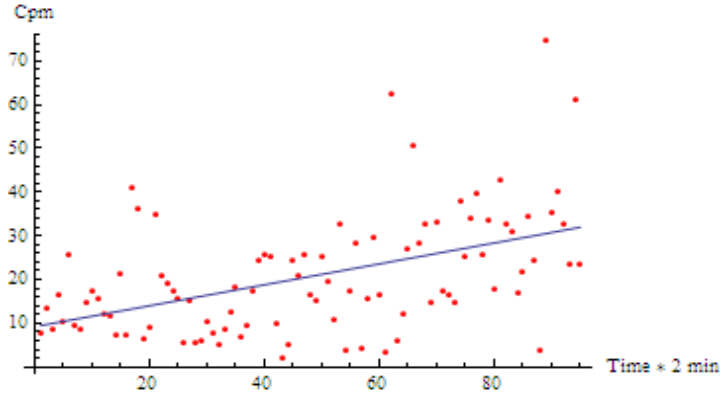

Figure 24. Data from a learning curve in Verrips 2015 with a highly gifted test subject and a single switch in conversation.

A model of complexity of codes

We may model complexity of codes for AC as in Figure 25. Here we assume that a code is related to all codes that consist of the same number of dits and dashes an to its inverse. So …. is only related to ---- and ..- is related to .-., to -.. and to --. A code with many related codes is considered more complex, both to remember and to retrieve.

This assumption is rather dubious because AC has structure, as shown in Figure 22, that is *not* taken into account. One might defend a table with a much lower average value because once you know, say, that l is in the j, k, l, m group, the code ..-. easily follows. The choice must be harder for b, u, v, w, p, comma, because they constitute three groups and there are six ways to combine two dits and two dashes, a consequence of Pascal's triangle usually taught in highschool. Whether these computed complexities are *psychological* reality to representative learners of either AC or CMC remains uncertain.

a 2	j 5	s 4
b 6	k 5	Sp 2
Bksp 2	l 5	t 4
c 2	m 5	u 6
d 2	n 2	v 6
e 2	o 2	w 6
f 4	p 6	x 5
g 4	q 5	y 5
h 4	r 4	z 5
i 2		

Figure 25. Complexities of codes according to formula comp [code] = 1 + # relatedcodes.

We ordered characters from the copy task reported above with AC and ordered from low pauses before the characters to high pauses before the characters and put the complexities underneath. Noting that in a copy task other phenomena play a role, and that the number of characters was low, we would expect higher estimated complexities to the right. Table 1 is significant according to the (GoodmannKruskalGammaTest).

This need not apply at any point of the learning curve, nor with every encoding or test subject. Or even for ourselves, With a frequency of 6 for b we would assume that somehow this code does not feel complex to ourselves. To verify, we did the same test with data from the experiment with CMC. As expected, here the data were neither suggestive nor significant. Psychology is not simple, alas.

c	n	SP	b	e	s	u	BS	a	p	d	g	h	o	i	z	j	r	w	m	l	v	t	k
2	2	2	6	2	4	4	2	2	6	2	4	4	2	2	5	5	4	6	5	5	6	4	5

Table 1. Characters entered with AC in copy task ordered from lower to higher pauses before them, with estimated complexities in the second row.

Char	No	Avg pause	SD	Complexity	Char	No	Avg pause	SD	Complexity
c	3	310.	52.3	2	o	18	554.4	477.6	2
n	48	351.2	183.9	2	i	28	654.3	383.5	2
SP	95	377.6	216.1	2	z	3	671.	532.4	5
b	6	421.	233.5	6	j	4	780.5	631.6	5
e	82	436.7	236.8	2	r	23	791.2	541.0	4
s	12	437.9	174.9	4	w	8	794.7	518.5	6
u	7	442.5	227.8	4	m	9	796.5	574.0	5
BS	27	483.3	288.6	2	l	14	799.5	411.7	5
a	33	494.2	430.0	2	v	14	845.7	484.7	6
p	6	507.3	364.4	6	t	10	979.7	513.2	4
d	26	529.7	399.3	2	k	10	989.8	458.6	5
h	10	546.7	294.5	4	Tot	552			

Table 2. Same data as in Table 1 but ordered on average pause before code and with more details. F, g, q, x, y were left out, frequency <= 2. Pauses are rather wide apart and SD's are large.

a 85	b 16	c 30	d 44	e 120	f 25	g 17	h 64
i 80	j 4	k 8	l 40	m 30	n 80	o 80	p 17
q 5	r 62	s 80	t 90	u 34	v 12	w 20	x 4
y 20	z 2	Sp 300	Bs 120	Total: 1489			

Figure 26. Frequencies of characters in printed Dutch text with 8% Backspaces and 20%

Sp.

We model lookup-time given complexity comp and x^{th} time this code is entered as

lookup-time = const * comp / (x + comp).

The value of the const must be in the same order of magnitude as the overhead, see below.

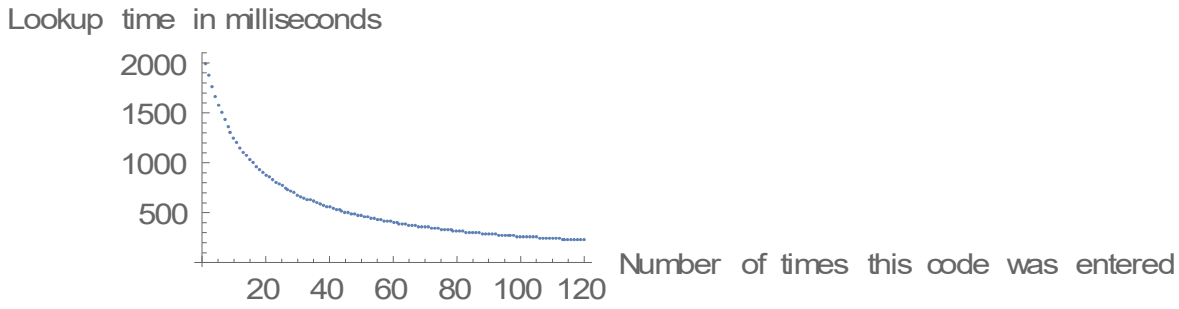

Figure 27. Model of lookup-time.

In figure 27 we took const = 2000 and comp = 15 milliseconds. With x = 1 we have 2000 * 15 / 16 = 1875 milliseconds and with x = 120 we have 2000 * 15 / 135 = 222 milliseconds. With comp 150 we would have needed x = 1200 to achieve the same rate enhancement.

If f [c, x] means that function f depends on character c, and on x, how often this character was entered, we have

Time-per-character [c, x] = codelength [c] + overhead + lookup-time [c, x]
lookup-time [c, x] = const * comp [c] / (x + comp [c])

In the following we will assume the value of const is equal to the value for overhead. If a certain character has a codelength of 1000 milliseconds, and overhead is 2000 milliseconds, we will start with a lookup-time of 2000 milliseconds and conclude 5000 milliseconds for this character or 60 / 5 = 12 characters per minute. If the lookup-time approaches zero with training we will approach 60 / 3 = 20 characters per minute. If overhead decreases and an output buffer forms rate must rise again we will come back on this later.

The graph in Figure 28 was computed based on the above and displays input rate as a function of training in number of characters per minute. Overhead is important, with overhead 1000 instead of 2000 milliseconds per characters we compute higher values. This effect is greater than a change in the assumed complexities or in the pause time.

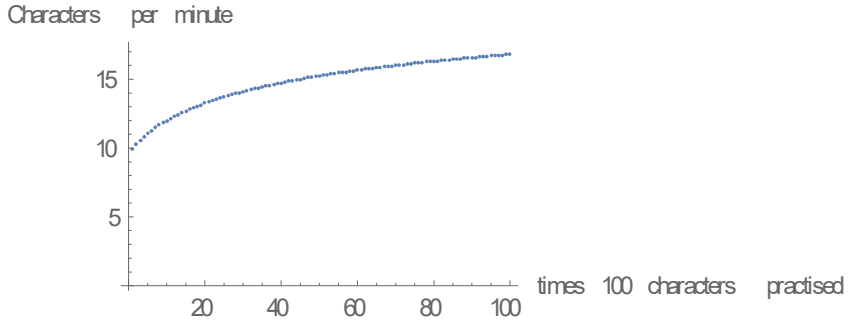

Figure 28. Characters per minute as a function of training accorded to computed model. Overhead = 2000 milliseconds.

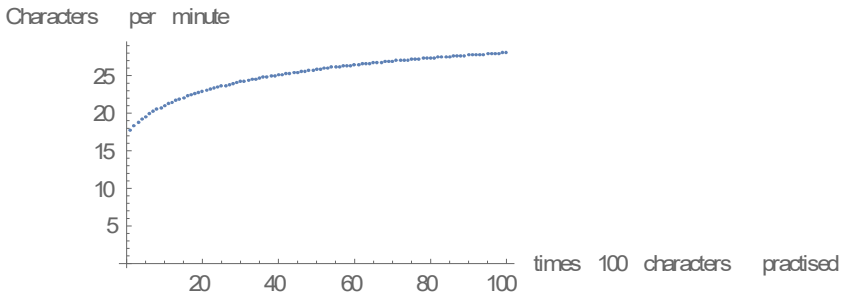

Figure 29. Characters per minute as a function of training. Overhead = 1000 milliseconds.

In both Figure 28 and Figure 29 horizontally 1 = 100 characters. As expected, after a few thousand characters input rate rises but slowly. Overhead time must slowly decrease with time but we should not assume it can ever approach zero. Figure 30 combines graphs with normal and low overhead as well as normal and lower complexities.

In these graphs we multiplied complexities with 10, assuming that one will not learn much from every code entered. On the long run, say after 100 000 codes, we will see but two graphs due to continued learning. The green graph in the middle represents a gradually diminishing overhead and we may imagine that on the long run it will cross the upper two graphs. And will approach an asymptote related to average code length, about one second per character with the switch we used[ix]. Though computed, these graphs have something speculative about them.

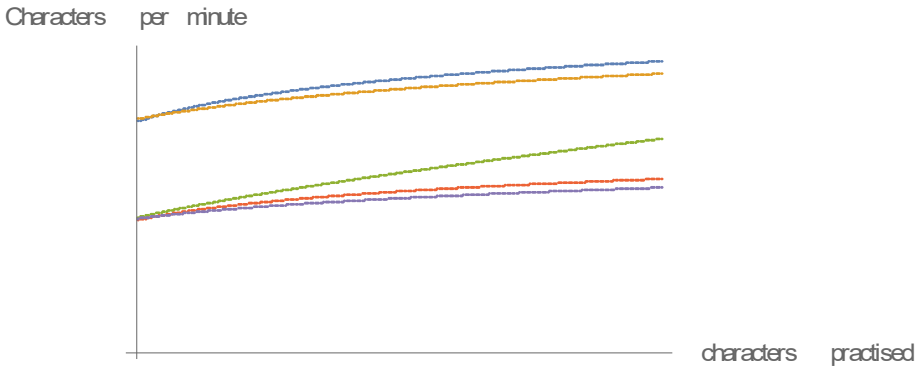

Figure 30. Computed graphs with varying overheads and complexities.

Interpretation

A sub-keystroke level model was presented of encoding with predictions of input speed that suggest a significant effect of attention and a limited effect of the complexity of some encoding. Learning was discussed superficially.

Discussion

Several models lend support for rather offering Alternative Code than Classical Morse Code. The question however is not settled and one wonders if it ever will be. Individual differences must play a part, both encodings are rather efficient and the difference must be modest.

Bias As the author acted as his own test subject to test his own software researcher bias is evident.

Pictures Pictures were made with Mathematica 10.2.

Data Notebooks with relevant data and computations as well as a pdf of the most important one with logdata is in www.depratendecomputer.nl/clicksandpauseswhenencoding.zip.

References

Baker A.E. and Hengeveld, K. 2013. Linguistics. Wiley and Blackwell.

Card, S, Moran, T, Newell, A. 1983. Psychology of Human Computer Interaction. Addison Wesley.

Kieras, D.E. 2005. Model Based Evaluation. In Jacko & Sears. The Human-Computer Interaction Handbook (2nd Ed). Mahwah, New Jersey: Lawrence Erlbaum Associates. Pp 1295-1314.

King, T. 1999. Morse Code in Rehabilitation and Education. Allyn & Bacon.

Kopland, R. 1999. Geluk is gevaarlijk.

MacKenzie, I. S. 2009. The one-key challenge: Searching for a fast one-key text entry method. Proceedings of the ACM Conference on Computers and Accessibility – ASSETS 2009, pp. 91-98.

MacKenzie, I.S. 2013. Human-Computer Interaction. An Empirical Research Perspective. Morgan Kaufman, Elsevier.

Moran, T.P. 1982. Too little, too late. ACM.

Moran, T.P. and Carroll, J.M, 1996. Wicked Problems. In Design Rationale: Concepts, Techniques, and Use, ed. Lawrence Erlbaum Associates. .

Patton, J. 2014. With Economy, P. User Story Mapping. O'Reilly.

Verrips, J. 2000. Test of a communication aid with stored text. International Journal of Rehabilitation Research. 169-175.

Verrips, J. 2012. www.depratendecomputer.nl/optimizingmorsecode.pdf.

Verrips, J. 2013. Comparison of oriented scanning and alternative code. Input rate of text with a single switch. Communication Matters Vol 27:2. 39-40.
www.depratendecomputer.nl/comparisonosandasm.pdf.

Verrips, J. 2013b. Conversations with a single switch.
www.depratendecomputer.nl/conversationswithasingleswitch.pdf.

Verrips, J. 2014. Volunteering. www.depratendecomputer.nl/volunteering.pdf.

Verrips, J. 2015. Design of WriteEasy.
www.depratendecomputer.nl/ConversationsWithWriteEasy.pdf.

Verrips, J. 2016. (draft). Volunteering with speaking software.
www.depratendecomputer.nl/morevolunteering.pdf.

i In the design of Alternative Code, first short codes were assigned to frequent characters (Sp, BS, e, n, a, i, d) and then related codes to consecutive characters. If only *one* click is different, in AC it will 'walk' from left to right as in f = .--, g = -.- and h = --., as in j = -... ,k = .-.., l = ..-. and m = ...- or as in numerals 1 to 5 1 = -.... etcetera. This last property was recently made more consistent. Some people prefer the display underneath, showing groupwise ordering of characters and codes. For the occasion, we also included codes for Function keys, ?? = change presentation, v?, change voice and ?x or hide codes, intended for candid use.

```
a  .-     b ..--    n ...    p .-.-  v .--.
i  -.     u --..    o ....    , -.-.  w -..-

c  ---    f .--     j  -...  r -..   x .---
d  --     g -.-     k  .-..  s .-.   y -.--
e ..      h --.     l  ..-.  t ..-   z --.-
                    m  ...-          q ---.

SP .      BS - BS* -_  Ret ---- Ret .......
?? .--..  v? ..--.     ?x  --.---

1 -....  2 .-... 3 ..-.. 4 ...-. 5 ....- 6 .....
1 _ 2 ._ 3.._ 4 ..._ 5 ..._ _ 6 ..._ _ _

F1 .-....  F2 ..-...  F3 ...-..  F4 ....-.
F7 --....  F8 -.-...  F9 -..-..  F10 -...-.
F13 ---... F1& .-_    F7& -._    F13& --_
```

ii Some defend a third switch to replace the pause.

iii “Een psalm” by Rutger Kopland.

iv This is not too amazing in so far as ten finger typists also rapidly adapt to keyboards with different layouts. But it does offer food for thought. As before I repeated groups of characters with related codes (like d = -.. u = ..- and r = .-. in CMC).

v It also helps to do the experiments one wants to do and to correct all errors. In the past minor changes of pause time, texts copied, training and motivation of had significant and repeatable effects. To do this reliably with members of the intended user groups is beyond us.

vi AC and CMC are in many respects comparable. Codes of length 4 with AC also have length 4 with CMC with the exception of BS. However, -___ also repeats BS in CMC so the awkward BS = ---- can not have had much effect. Likewise for most codes of length 3.

vii $1/(x^{0.5}+1)$ appears even better than 1/(x+1) but was not used.

viii In Wolframese code is succinct:

```
speed [x_,a_,b_,c_,d_] := If [x<31, (a x +b)/(c x + d), speed [x-30,a,b,c,d] + speed [30,a,b,c,d]]
picture [a_:1,b_:0,c_:1,d_:5] := ListPlot [Table[speed[x,a,b,c,d],{x,1,120}],Ticks->None, AxesLabel ->{"Time spent practicing","Input speed"}]
picture [ ]  (* default values are used, therefore this means picture[ 1, 0, 1, 5] *)
```

ix Professional telegraphists used special switches and achieved much higher rates, often 40 *words* per minute.