I. SCOTT MACKENZIE
KUMIKO TANAKA-ISHII

TEXT ENTRY SYSTEMS

Mobility, Accessibility, Universality

MORGAN KAUFMANN

Text Entry Systems: Mobility, Accessibility, Universality

Edited by

I. Scott MacKenzie
and
Kumiko Tanaka-Ishii

MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER SCIENCE

AMSTERDAM BOSTON LONDON NEW YORK
OXFORD PARIS SAN DIEGO SAN FRANCISCO
SINGAPORE SYDNEY TOKYO

Publisher Diane Cerra
Publishing Services Manager George Morrison
Project Manager Mónica González de Mendoza
Editorial Assistant Asma Palmeiro
Editorial Intern Marisa Crawford
Cover Design Hannus Design
Composition Charon Tec
Copyeditor Valerie Koval
Proofreader Charon Tec
Indexer Charon Tec
Interior printer Maple-Vail Book Manufacturing Group
Cover printer Phoenix Color

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
MacKenzie, I. Scott, 1951-
Text entry systems : mobility, accessibility, universality / I. Scott MacKenzie, Kumiko Tanaka-Ishii.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-12-373591-1 (pbk. : alk. paper)
ISBN-10: 0-12-373591-2 (pbk. : alk. paper) 1. Computers. 2. Electronic data processing—Data entry. 3. Human-computer interaction. 4. Natural language processing (Computer science) I. Tanaka-Ishii, Kumiko. II. Title.
QA76.5M18747 2007
004—dc22

2006037436

ISBN 13: 978-0-12-373591-1
ISBN 10: 0-12-373591-2

Printed in the United States of America
07 08 09 10 11 5 4 3 2 1

3

CHAPTER

Measures of Text Entry Performance

Jacob O. Wobbrock University of Washington, Seattle, WA, USA

3.1 INTRODUCTION

When people encounter a new text entry method, the first question they usually ask is, "How fast is it?" Although speed is a key feature of any text entry method, there is much more that can and should be said. Clearly, accuracy is important and is not entirely independent of speed. Even so, there are different kinds of accuracy, such as accuracy *during* input, when errors may be made and corrected, and accuracy *after* input, when the entered text is considered complete. But do speed and accuracy tell us everything? What can we say about how learnable a method is, or how efficient? What measures are applicable to keyboard typing or stylus-based gestures? People are sometimes surprised at how much can be said about the performance of a text entry method. Certainly, we can capture and quantify a great deal more than just speed and accuracy.

This chapter presents numerous empirical measures of text entry performance. These measures are, for the most part, *method-agnostic,* applicable to a variety of text entry methods. Speed is one example of a method-agnostic measure. Other measures presented near the end of the chapter are *method-specific,* inherent to keyboard typing, selector movement, or gestural methods.

Although this chapter does not focus on experimental or methodological issues (see Chap. 4), it is important to note that the meaning of an empirical measure is dependent upon the manner in which the text entry data are captured. For example, entry rates must be regarded differently when one is composing text versus copying it, since the former may involve substantial thinking time and other considerations that are difficult to measure. Therefore, copying text is usually preferred when evaluating text entry methods in laboratory settings (MacKenzie & Soukoreff, 2002a). Even for copying text, however, some studies have required participants to maintain synchronicity with the source text, disallowing incorrect characters (Venolia & Neiberg, 1994; Isokoski & Kaki, 2002; Evreinova *et al.,* 2004; Ingmarsson *et al.,* 2004), while others have disabled backspace and other error correction mechanisms (Matias *et al.,* 1996; MacKenzie & Zhang, 1999). As we might imagine, these rather artificial limitations may adversely affect participants' entry and error rates.

Owing to its naturalness and widespread use, the *unconstrained text entry evaluation paradigm* (Soukoreff & MacKenzie, 2001, 2003; Wobbrock & Myers, 2006d) is the experimental methodology assumed by most measures in this chapter. In these evaluations, a short *presented string* is shown to participants who enter it using the method under investigation. A participant's entered result is called the *transcribed string*. This process is repeated over many trials. During transcription, all printable characters are accepted as legitimate entries, and backspace is used as the sole means of error correction. No error beeps or other intrusions affect the fluid entry of text, and participants are simply told to "proceed quickly and accurately" (Soukoreff & MacKenzie, 2003) in an effort to balance speed and accuracy without unduly favoring one over the other. Log files are written that provide a record of the *input stream,* which is the sequence of actions (i.e., keystrokes, gestures, and so on) taken by participants during transcription. This input stream is then parsed and interpreted according to the measures presented in the rest of this chapter.

3.2 AGGREGATE MEASURES

Aggregate text entry measures refer to those that characterize a method's performance *on the whole.* These are contrasted with character-level measures, explained in Section 3.3. In particular, this section focuses on method-agnostic aggregate measures, mainly dealing with entry rates, error rates, and various measures of efficiency.

3.2.1 Entry Rates

Although characterizing the entry rate of a text entry method is mostly straightforward, a few nuances and alternatives exist that warrant some discussion. This section reviews various speed-related measures and highlights their differences.

Words per Minute (WPM)

Words per minute (WPM) is perhaps the most widely reported empirical measure of text entry performance. Since about 1905, it has been common practice to regard a "word" as 5 characters, including spaces (Yamada, 1980)[1]. Importantly, the WPM measure does not consider the number of keystrokes or gestures made *during* entry,

[1] Note that different text entry communities have different definitions of many common terms, including the term "word." For example, speech-based text entry researchers define "words" as actual spoken words, not 5 characters. See Chap. 8 for a discussion of speech-based text entry.

but only the length of the resulting transcribed string and how long it takes to produce it. Thus, the formula for computing WPM is:

$$\mathrm{WPM} = \frac{|T| - 1}{S} \times 60 \times \frac{1}{5}. \tag{3.1}$$

In Eq. (3.1), T is the final transcribed string entered by the subject, and $|T|$ is the length of this string. As stated, T may contain letters, numbers, punctuation, spaces, and other printable characters, but not backspaces. Thus, T does not capture the text entry *process,* but only the text entry *result.* The S term is seconds, measured from the entry of the first character to the entry of the last, including backspaces. The "60" is seconds per minute and the "1/5" is words per character.

Sometimes researchers prefer to report entry rates in characters per second (CPS). Although this is less common than WPM, the measure is a simple transformation of WPM, since the first ratio in Eq. (3.1) is in CPS.

The "-1" in the numerator deserves special mention and is often neglected by researchers when reporting entry rates. As stated, the S term in Eq. (3.1) is measured from the entry of the first character to the entry of the last. Consider this example from MacKenzie (2002b):

```
the quick brown fox jumps over the lazy dog (43 chars)
^                                         ^
t = 0 sec                                 t = 20 sec
```

It would be incorrect to calculate CPS as 43/20, since timing begins with the entry of the first character. Thus, the proper CPS calculation is $(43 - 1)/20$. Of course, multiplying this ratio by $60 \times 1/5$ gives us WPM.

Adjusted Words per Minute (AdjWPM)

In the unconstrained text entry evaluation paradigm, participants can enter text freely and must therefore strike a balance between speed and accuracy[2]. From Eq. (3.1), it is clear that errors made and then corrected *during* text entry reduce WPM, since it takes time to make and correct errors and WPM considers only the length of the transcribed string. In contrast, errors *remaining* in the transcribed string, so-called *uncorrected errors,* are at odds with speed, since participants can go faster if they leave more errors. A way of penalizing WPM in proportion to the number of uncorrected errors is to compute an adjusted entry rate (Matias *et al.,* 1996; Wobbrock *et al.,* 2006):

$$\mathrm{AdjWPM} = \mathrm{WPM} \times (1 - U)^{a}. \tag{3.2}$$

[2] To help participants in this regard, experimenters may suggest that participants enter text "as if they were writing an e-mail to a colleague" or other such guideline.

In Eq. (3.2), U is the uncorrected error rate ranging from 0.00 to 1.00, inclusive. The variable a might be called the "penalty exponent" and is usually set to 1.0, but may be increased in order to more severely penalize WPM in light of uncorrected errors. For example, if we have 10% uncorrected errors, our adjusted WPM will be 90% of the raw WPM if $a = 1.0$, but only 81% of the raw WPM if $a = 2.0$. For an example of using AdjWPM in a longitudinal study, see prior work (Wobbrock *et al.,* 2006).

Because of the rather arbitrary nature of AdjWPM, some experimenters (Lewis, 1999) have insisted that participants correct *all* errors during entry, thereby eliminating uncorrected errors and enabling WPM to be a single measure of performance. In these studies, trials that contain uncorrected errors may be discarded and replaced with additional trials. Such trials should, however, be reported separately as a percentage of all trials performed.

Keystrokes per Second (KSPS)

As noted, the calculation of WPM and AdjWPM do not take into account what happens *during* text entry. To do this, it is also useful to view text entry activity as a process of data transfer from the user to the computer. We can then characterize the "data rate." For example, some speech recognition systems may enter long chunks of text very quickly, but also make many errors. These systems often support rapid error correction with a designated phrase such as "scratch that." Users may retry certain phrases many times before the system correctly recognizes them, but WPM will not capture all of this activity. Instead, we can use keystrokes per second (KSPS) according to the following formula:

$$\mathrm{KSPS} = \frac{|IS| - 1}{S}. \tag{3.3}$$

In Eq. (3.3), IS is the input stream, which contains all entered characters, including backspaces. As in Eq. (3.1), the timing for S in seconds is assumed to begin with the entry of the first character and end with the entry of the last. The use of the word "keystrokes" in this measure should not be taken strictly, as the measure is still applicable to nontyping text entry methods[3]. (The term might instead be "bytes per second" (BPS), but in many of today's character sets, characters are no longer single bytes, so this term is not preferred.)

As an example, consider the following input stream in which "<" symbols indicate backspaces and transcribed letters are in bold:

```
tha<e p<quik<ck brwn<<own t<fox (31 keystrokes)
^                                ^
t = 0 sec                        t = 10 sec
```

[3] For historical reasons and because of its relationship to the *keystrokes per character* (KSPC) performance measure (see Section 3.2.2), KSPS uses the word "keystrokes" to refer to any token in the input stream, whether it was generated by a literal keystroke or not. Although numerous text entry methods are *not* keystroke-based methods, they still generate an input stream of characters, spaces, and backspaces, and thus KSPS applies just fine.

This input stream results in the transcribed string "the quick brown fox". The proper KSPS calculation for this input stream is $(31 - 1)/10 = 3.0$. Note that this differs from a calculation of CPS: $(19 - 1)/10 = 1.8$.

Using KSPS, researchers can estimate an "empirical upper bound" by assuming that all entered characters are correct. Such a calculation tells us how fast a method might be (in CPS) if its accuracy during entry were improved to 100%. For an example use of KSPS (referred to as BPS), see prior work (Wobbrock *et al.,* 2004).

Gestures per Second (GPS)

If KSPS is the data rate of a method, then gestures per second (GPS) is the "action rate." At first blush, the term "gestures" may seem applicable only to stroke-based text entry, but it should be interpreted more broadly. Here, a "gesture" is an atomic action taken during the text entry process. What constitutes an "atomic action" is dependent upon the method under investigation. For a stroke-based method, it would be any individual unistroke. For a stylus keyboard, it would be a stylus tap. For a physical keyboard, it would be an attempt to strike a key. For a speech recognition system, it would be an utterance. All of these are examples of gestures. Notably, the actions themselves do not have to be text-producing. For example, tapping a sticky SHIFT key on a stylus keyboard may be consider a gesture, even though SHIFT itself does not produce a character. Since different methods will require different gestures, it is important that researchers who use this measure report exactly what they consider a gesture to be.

The GPS measure tells us how fast the user is taking actions. We also therefore introduce the concept of a "nonrecognition," which we define as "any unproductive gesture." This may be an unrecognized unistroke, a stylus tap on "dead space" between virtual keys, a missed physical key, or an unrecognized utterance. We will represent nonrecognitions as "ø". With this understanding, we can define GPS as:

$$\mathrm{GPS} = \frac{|IS_{\varnothing}| - 1}{S}. \tag{3.4}$$

In Eq. (3.4), $IS_{ø}$ stands for the input stream *including all actions.* Text entry experiments can capture these actions by adding them to performance logs. For physical gestures like striking a key, video analysis can be used to discover actions, including nonrecognitions (i.e., missed keys).

Consider this example, in which "ø" represents a nonrecognition and "•" represents SHIFT:

```
•Tha<øe p<quiøk<ck brwn<<own t<foxø (35 gestures)
^                                 ^
t = 0 sec                         t = 10 sec
```

The proper GPS calculation is $(35 - 1)/10 = 3.4$. It should now be clear that $\mathrm{GPS} \geq \mathrm{KSPS} \geq \mathrm{CPS}$ for a given text entry trial.

Learning Curves

Another common use of entry rates is to graph WPM over time and model the points according to the power law of learning (Card *et al.*, 1983)[4]. Such models take the following form:

$$\text{WPM} = aX^b. \tag{3.5}$$

In Eq. (3.5), X is the variable "time," usually a session number, and a and b are fitted regression coefficients. The value for a is "initial performance" and the value for b, often below 0.5, determines the steepness of the curve. The result of fitting such curves allows us to estimate performance in future sessions, especially if the goodness of fit (R^2) is high. For additional discussion, see the following examples: MacKenzie and Zhang (1999), Isokoski and MacKenzie (2003), Lyons *et al.* (2004), Wobbrock and Myers (2006a), and Wobbrock *et al.* (2006).

3.2.2 Error Rates

It may come as no surprise that measuring error rates in unconstrained text entry experiments is trickier than measuring entry rates. There are multiple error metrics, but most are concerned with the distinction between errors *during* entry ("corrected errors") and errors *after* entry ("uncorrected errors").

Keystrokes per Character (KSPC)—Performance Measure

One way to quantify errors during text entry is to use the keystrokes per character (KSPC) performance measure (Soukoreff & MacKenzie, 2001)[5]. This measure is a simple ratio of the number of entered characters (including backspaces) to the final number of characters in the transcribed string. The formula for calculating KSPC is:

$$\text{KSPC} = \frac{|IS|}{|T|}. \tag{3.6}$$

[4] See Chap. 4, Section 4.4, for experimental issues concerning learning curves and longitudinal studies.

[5] This is not to be confused with the KSPC *characteristic measure* (see Sections 3.2.3 and 5.2), which describes the inherent efficiency of a method as the number of key or button presses required to generate a character. Note that like KSPS, the KSPC performance measure does not require a keystroke-based method. All that is required to use this measure effectively is a text entry method that generates a character-level input stream containing tokens like letters, spaces, and backspaces.

As in Eq. (3.3), IS includes all characters, including backspaces. To use our previous example:

```
tha<e p<quik<ck brwn<<own t<fox (31 keystrokes)
```

The transcribed string is “the quick brown fox”, which contains 19 characters. Thus, our KSPC accuracy ratio is 31/19 = 1.63. Note that lower KSPC is better and 1.0 represents “perfect entry.” For an example use of KSPC, readers are referred to prior work (Wobbrock *et al.*, 2003b).

A limitation of the KSPC performance measure is that it makes no distinction between backspaced characters that were initially correct and backspaced characters that were initially incorrect (Soukoreff & MacKenzie, 2004). In the example above, the first “wn” in “brwn” were, in some sense, correct, but erased in order to enter the omitted “o”. The “wn” are treated as any other incorrect letters, even though they were entered correctly by the participant. Much more complicated procedures are needed to correctly differentiate corrected-and-wrong and corrected-but-right characters. Such procedures require a *character-level* error analysis of the input stream (see Section 3.3.3).

Gestures per Character (GPC)

An extension of KSPC is gestures per character (GPC). As with GPS, we regard a “gesture” as any atomic action taken during the text entry process, and a “nonrecognition” to be any unproductive gesture. Researchers should report GPS and GPC with clear definitions of what they consider a gesture. The GPC measure gives us an indication of the accuracy during entry *including futile actions*. The formula for GPC is:

$$\text{GPC} = \frac{|IS_{\varnothing}|}{|T|}. \tag{3.7}$$

In Eq. (3.7), $IS_{\varnothing}$ encodes all characters, backspaces, nonrecognitions, and other actions. To use our previous example in which “•” represents SHIFT:

```
•Tha<∅e p<qui∅k<ck brwn<<own t<fox∅ (35 gestures)
```

The transcription is “The quick brown fox”. Thus, GPC is 35/19 = 1.84. That is, nearly two gestures (i.e., actions) were required for every character that ended up in the transcribed string. Thus, GPC is also some measure of efficiency, since entry methods that require more actions per character will have a higher GPC, even if the accuracy of participants taking those actions is reasonable. For an example use of GPC, see prior work (Wobbrock *et al.*, 2003b).

Minimum String Distance (MSD)

Thus far, we have considered accuracy during text entry, but what about the accuracy of the resultant transcribed string? For this, we may use the *minimum string distance*

```
MSD($P$, $T$)
 1  $D \leftarrow$ **new** matrix of dimensions $|P| + 1, |T| + 1$
 2  **for** $i \leftarrow 0$ **to** $|P|$ **do**
 3      $D[i, 0] \leftarrow i$
 4  **for** $j \leftarrow 0$ **to** $|T|$ **do**
 5      $D[0, j] \leftarrow j$
 6  **for** $i \leftarrow 1$ **to** $|P|$ **do**
 7      **for** $j \leftarrow 1$ **to** $|T|$ **do**
 8          $D[i, j] \leftarrow \textsc{Min}(D[i-1, j] + 1,$
 9                           $D[i, j-1] + 1,$
10                           $D[i-1, j-1] + P[i-1] \neq T[j-1])$
11  **return** $D[|P|, |T|]$
```

FIGURE 3.1 Algorithm for computing the minimum string distance. In this case, the ≠ comparison returns integer "1" if **true** and integer "0" if **false**. Adapted from Soukoreff and MacKenzie (2001) and Wobbrock and Myers (2006d).

statistic (Levenshtein, 1965; Wagner & Fischer, 1974; Soukoreff & MacKenzie, 2001). The MSD statistic gives us the "distance" between two strings in terms of the lowest number of error-correction operations required to turn one string into the other. The error-correction operations available to us are *inserting* a character, *deleting* (or *omitting*) a character, and *substituting* a character (Damerau, 1964; Morgan, 1970; Gentner *et al.*, 1984). Consider the following presented (P) and transcribed (T) strings:

```
P: the quick brown fox
T: tha quck browzn fox
```

Manual inspection shows us that we have a substitution of "a" for "e", an omission of "i", and an insertion of "z". So for this example, MSD = 3. However, sometimes the MSD result is not so clear:

```
P: quickly
T: qucehkly
```

What is the MSD in this case? Fortunately, an algorithm exists to compute the MSD for us (Fig. 3.1). As parameters, the algorithm takes the presented and transcribed strings.

Using the MSD algorithm, we discover that "quickly" and "qucehkly" can be equated with no fewer than three operations[6]. We can compute an MSD error rate according to the following equation (Soukoreff & MacKenzie, 2001):

$$\text{MSD error rate} = \frac{\text{MSD}(P, T)}{\text{MAX}\left(|P|, |T|\right)}. \tag{3.8}$$

[6] Determining *which* three operations is another matter, requiring a character-level error analysis. See Section 3.3.2.

Correct (C)	All correct characters in the transcribed text.
Incorrect-not-fixed (INF)	All incorrect characters in the transcribed text.
Incorrect-fixed (IF)	All characters backspaced during entry.
Fix (F)	All backspaces.

TABLE 3.1 Character classes used in computing corrected, uncorrected, and total error rates.

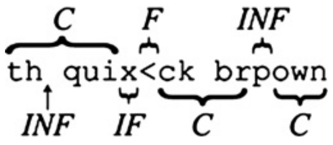

FIGURE 3.2 An example phrase showing character classifications according to the four categories in Table 3.1. Adapted from Soukoreff and MacKenzie (2003). Used with permission.

As Soukoreff and MacKenzie (2001) explain, the denominator uses the longer of the two strings so that (a) the measure does not exceed 1.00, (b) participants do not receive undue credit for entering less text than contained in the presented string, and (c) an appropriate penalty is incurred if more text is entered than is contained in the presented string. Thus, for "quickly" and "qucehkly", our error rate is $3/8 = 0.375$.

Corrected, Uncorrected, and Total Error Rates

One drawback of using KSPC for accuracy during entry and MSD for accuracy after entry is that they are not easily combined. Recent work has therefore set out to devise a unified error metric. The result was the development of corrected, uncorrected, and total error rates (Soukoreff & MacKenzie, 2003). These three error rates depend on classifying all entered characters into one of four categories (Table 3.1).

Consider the classification example shown in Fig. 3.2. Automating these classifications is straightforward because we do not need to know *which* characters belong to each category, but simply *how many* characters do. Characters belonging to the F and IF classes can be counted by making a backward pass over the input stream. The size of the INF class is equal to $\mathrm{MSD}(P, T)$. The size of the C class equals $\mathrm{MAX}(|P|, |T|) - \mathrm{MSD}(P, T)$.

Similar to the KSPC performance measure, the IF class contains *all* backspaced characters, regardless of their initial correctness. In Fig. 3.2, if the "x" were a "c" it would still belong to the IF class. This is one limitation of these aggregate error metrics that is remedied by character-level error analyses (see Section 3.3.3).

Using these character classes, we can define our three error rates. The corrected error rate supersedes the KSPC performance measure. It is calculated with Eq. (3.9):

$$\text{corrected error rate} = \frac{\mathrm{IF}}{\mathrm{C} + \mathrm{INF} + \mathrm{IF}}. \tag{3.9}$$

The uncorrected error rate supersedes the MSD error rate and is calculated using Eq. (3.10):

$$\text{uncorrected error rate} = \frac{\text{INF}}{\text{C} + \text{INF} + \text{IF}}. \tag{3.10}$$

The total error rate is merely the sum of Eqs. (3.9) and (3.10):

$$\text{total error rate} = \frac{\text{IF} + \text{INF}}{\text{C} + \text{INF} + \text{IF}}. \tag{3.11}$$

In practice, it is common for corrected errors to vastly outnumber uncorrected errors. This can make the total error rate rather uninformative, since it will mostly reflect corrected errors. Furthermore, evaluators must be careful to point out that higher corrected errors decrease speed, but higher uncorrected errors increase speed. A rapid method that creates many corrected errors, has efficient error correction, and leaves few uncorrected errors can still be considered a successful method, since it produces accurate text in relatively little time.

Cumulative and Chunk Error Rates

Thus far, we have considered error metrics from the unconstrained text entry evaluation paradigm. However, many user test software programs and typing tutors prior to 2001 neither used the unconstrained paradigm nor analyzed errors in this way. Instead, many of these programs (Matias *et al.,* 1996) disallowed error correction all together and simply treated an error as any character out of sync with the presented text. This gave rise to what has been called the *cumulative error rate.* Consider again our example:

```
P: the quick brown fox
T: tha quck browzn fox
     x   xxxxxxxx
```

In this case, 19 characters were presented, and of those entered, 9 disagree in a pairwise comparison. Thus, the cumulative error rate is 9/19 = 47.4%.

Clearly, however, the second block of errors—the eight-error "chunk"—was precipitated by the omitted "i" in "quck". Thus, Matias *et al.* (1996) defined the *chunk error rate* to account for this by treating error chunks as single errors, thus permitting a more realistic automated analysis. The chunk error rate for the above example would be 2/19 = 10.5%. However, we can see a drawback of the chunk error rate, since a third error has been engulfed by the same chunk: the inserted "z" in "browzn" deserves to be counted as a separate error, but the chunk error rate would fail to do so. At this point, we can appreciate the development of the previous error rates designed to accommodate the unconstrained text entry evaluation paradigm.

3.2.3 Efficiency Measures

Various measures also exist that describe a method's efficiency. The first of these presented in this section is a characteristic (or "model") measure. The others are empirical measures based on text entry experiments.

Keystrokes per Character (KSPC)—Characteristic Measure

The KSPC *characteristic measure* (MacKenzie, 2002c) quantifies how many key presses are required to generate a single character in a text entry method. This is not the same as the KSPC performance measure (see Section 3.2.2), which is a ratio of entered characters to transcribed characters (Soukoreff & MacKenzie, 2001). As before, the term "keystrokes" should be interpreted broadly. For example, a hypothetical stroke-based stylus method that requires two distinct unistrokes to generate each character would have a characteristic KSPC of 2.0.

MacKenzie (2002b) provides numerous examples to illustrate the calculation of the KSPC characteristic measure. Entering lowercase letters on a standard Qwerty keyboard has a KSPC of 1.0000, since each key corresponds to a single letter. The multitap method used on most mobile phones uses only eight keys for letters (keys 2–9) and a ninth key (often the 0 key) for SPACE. Its KSPC is 2.0342. The disambiguating phone keypad technique *T9* (www.tegic.com) resolves phone key sequences into their most likely words without requiring multiple presses of the same key. Its KSPC is 1.0072. Word prediction and completion techniques have KSPCs under 1.0000 since multiple letters may be entered in single actions (e.g., tapping a candidate word with a stylus).

How are these figures for KSPC calculated? The answer depends on the type of method being considered, but all calculations require a *language model.* For our purposes, a language model contains single letters, letter pairs, or entire words along with a count of how many times those entities were observed in a corpus of text (Table 3.2).

When the entry of letters is independent of previously entered letters, the "letter model" will do. For each letter in the model, we append the number of keystrokes required to generate that letter in the text entry method being considered. The formula for the KSPC characteristic measure is:

$$\text{KSPC} = \frac{\sum_{c \in C} (K_c \times F_c)}{\sum_{c \in C} F_c}. \qquad (3.12)$$

In Eq. (3.12), c is a single character in the letter model C, K_c is the number of keystrokes required to enter c, and F_c is the frequency count for c.

When the entry of a letter depends on the previously entered letter, we must use the letter-pair model. (Letter pairs are also sometimes called letter digraphs or letter

Letter model	Letter-pair model	Word model
_ 90563946	e_ 18403847	the 6187925
a 32942557	_t 14939007	of 2941789
b 6444746	th 12254702	and 2682874
c 12583372	he 11042724	to 2560344
d 16356048	s_ 10860471	a 2150880

TABLE 3.2 Examples of three language models used to calculate KSPC. Underscores ("_") represent SPACE. Frequency counts are from the British National Corpus. Adapted from MacKenzie (2002c).

digrams.) Selection keyboards that allow a user to move a selector over an on-screen depiction of keys with successive presses of directional keys require the letter-pair model, since the number of key presses required for each letter depends on the letter from which we start (i.e., the previous letter). In this case, Eq. (3.12) still applies, but c is now a letter-pair and C contains all $27 \times 27 = 729$ letter-pairs in the model.

When the result of a key press depends on word-level context, the word model must be used to calculate KSPC. To each word and frequency count in the model (Table 3.2), we append the number of keystrokes required to generate that word, including a trailing space. The KSPC calculation is then:

$$\text{KSPC} = \frac{\sum\limits_{w \in W} (K_w \times F_w)}{\sum\limits_{w \in W} (C_w \times F_w)}. \tag{3.13}$$

In Equation (3.13), w is a word in the word model W, K_w is the number of keystrokes required to enter w, F_w is the frequency count for w, and C_w is the number of characters in w. Both K_w and C_w assume the presence of a trailing space after the word.

The KSPC characteristic measure is one model-based way of quantifying the efficiency of a text entry method. The following sections describe other measures of text entry efficiency based on text entry experiments.

Correction Efficiency

The efficiency of error correction can be captured with the *correction efficiency* metric (Soukoreff & MacKenzie, 2003). This metric is defined using the character classes from Section 3.2.2. Correction efficiency is defined as:

$$\text{correction efficiency} = \frac{\text{IF}}{\text{F}}. \tag{3.14}$$

As Eq. (3.14) shows, the correction efficiency metric is the ratio of incorrect-fixed characters to the fix keystrokes that correct them. For basic keyboard typing, this metric will be 1.00, since characters can be created and erased in single key presses. However, other methods may allow multiple characters to be erased with a single fix, such as the Fisch in-stroke word completion technique (Wobbrock & Myers, 2006b), which allows word suffixes to be erased with a special backspace stroke. As a second example, consider speech recognition systems, which often allow multiple words to be erased with a single command like "scratch that," which could be considered a single fix "keystroke."

Participant Conscientiousness

Although it is not an efficiency metric per se, *participant conscientiousness* is related to correction efficiency (Soukoreff & MacKenzie, 2003). The metric compares the number of errors corrected to the total number of errors made, indicating how meticulous participants were in correcting errors as they entered text. The formula for participant conscientiousness is:

$$\text{participant conscientiousness} = \frac{\text{IF}}{\text{IF} + \text{INF}}. \tag{3.15}$$

Utilized and Wasted Bandwidth

The KSPS data rate (see Section 3.2.1) viewed text entry as the process of data transfer from the user to the computer. Along with the KSPS data rate, we can characterize the *utilized bandwidth* of a method (Soukoreff & MacKenzie, 2003)—the proportion of transmitted keystrokes that contribute to the correct aspects of the transcribed string:

$$\text{utilized bandwidth} = \frac{\text{C}}{\text{C} + \text{INF} + \text{IF} + \text{F}}. \tag{3.16}$$

Accordingly, the *wasted bandwidth* is defined as its complement:

$$\text{wasted bandwidth} = \frac{\text{INF} + \text{IF} + \text{F}}{\text{C} + \text{INF} + \text{IF} + \text{F}}. \tag{3.17}$$

Cost per Correction (CPC)

To more finely characterize error correction, we can use the *cost per correction* metric (Gong & Tarasewich, 2006). This metric was developed to capture the relative cost of error corrections in light of the fact that some errors go unnoticed until after a few additional characters have been entered. Consider these three input streams in which correction sequences have been grouped in parentheses, spaces have been added for clarity, and erroneous letters have been made bold:

```
(pui<<<) quick
(p<) q (r<r<) uick
(p<) q (v<) u (j<) ick
```

Gong and Tarasewich (2006) term the above sequences in parenthesis *corrections*. A single correction[7] consists of consecutive chunks of backspaced letters and the backspaces that remove them. In the first sequence, one error, the "p", inspires the removal of the following "ui", resulting in a single correction block of six keystrokes. In the second sequence, the "p" is an initial error immediately corrected, but two attempts at "u" result in "r"; this sequence therefore has two correction blocks. The third sequence contains three separate correction blocks separated by transcribed characters.

Using the notion of a "correction" and the keystroke classes defined previously (see Section 3.2.2), cost per correction (CPC) is defined as:

$$\text{CPC} = \frac{\text{IF} + \text{F}}{\text{corrections}}. \tag{3.18}$$

Applying Eq. (3.18)[8] to the first sequence results in a CPC of $(3 + 3)/1 = 6$. For the second sequence, we have $(3 + 3)/2 = 3$. For the third sequence, we have $(3 + 3)/3 = 2$. These results correspond to intuition, which suggests that the "damage done" by the first sequence's error is greater than that of the second or third sequence.

3.3 CHARACTER-LEVEL MEASURES

Thus far, we have considered measures that produce results for an entire input stream or transcribed string. Importantly, many of our measures have utilized only *counts of letters* (e.g., number of backspaced letters) without regard to *what those letters are*. For example, the IF character class may contain numerous already-correct characters that have been either mistakenly or purposefully erased (Soukoreff & MacKenzie, 2004). For finer grained analyses that differentiate among individual characters, we must use *character-level analyses*. Such analyses are generally more complicated than

[7] Although Gong and Tarasewich (2006) use the term "correction," I find this term easily confused with a single backspace. Instead, I prefer the term "correction sequence" or "correction block."

[8] Gong and Tarasewich (2006) include a third term in the numerator that represents "control keystrokes" used to produce any letters in correction blocks. I omit this term because the example sequences are method-agnostic and devoid of any method-specific control keys (e.g., NEXT in Multitap).

aggregate analyses, particularly in the case of errors. This section gives an overview of method-agnostic character-level measures.

3.3.1 Intra- and Intercharacter Time

The character-level measure somewhat analogous to entry rate (see Section 3.2.1) is *intracharacter time,* or the time to make a single character. Exactly how this value is measured will depend on the type of method under investigation, but most text entry methods support some notion of this measure. In a unistroke method, for example, the intracharacter time is the time from *pen down* to *pen up*. This measure has been used to compare unistroke character speeds in Graffiti and EdgeWrite (Wobbrock *et al.,* 2003b). For a continuous-stroking method like Quikwriting (Perlin, 1998), the intracharacter time is from when the stylus exits and returns to the center "hub" area. In a typing-based method, we could measure times from *key down* to *key up,* although as others have pointed out (Rumelhart & Norman, 1982), the full time to strike a key must include the time to move to the key before it can be depressed. The use of high-speed video can make it possible to identify the intracharacter times involved in keyboard typing, although this is a labor-intensive process.

The formula for calculating the average intracharacter time from a time-stamped input stream *IS* is

$$\text{intracharacter time} = \frac{\sum_{i=1}^{|IS|} \left(\text{character-end}_i - \text{character-start}_i\right)}{|IS|}. \tag{3.19}$$

In Eq. (3.19), the character-start$_i$ and character-end$_i$ values are time stamps for the *i*th character in the input stream. The first character in *IS* is at index $i = 1$.

Accordingly, we can also define the *intercharacter time,* which tells us how long participants delay between the end of one character and the beginning of the next. For example, world-champion typists, who can type up to 200 WPM, have average interkeystroke intervals of about 60 ms, which is close to the neural transmission time between the brain and fingers (Rumelhart & Norman, 1982). The intercharacter time can also be useful in spotting learning or recall difficulties in gestural alphabets, since novices may pause at length when trying to remember how to make certain characters (Wobbrock *et al.,* 2005a).

The formula for the average intercharacter time is

$$\text{intercharacter time} = \frac{\sum_{i=2}^{|IS|} \left(\text{character-start}_i - \text{character-end}_{i-1}\right)}{|IS| - 1}. \tag{3.20}$$

Note that Eq. (3.20) requires that we have at least two input stream characters.

```
ALIGN ($P$, $T$, $D$, $x$, $y$, $P'$, $T'$, **ref** *alignments*)
**if** $x = 0$ **and** $y = 0$ **then**
  *alignments* $\leftarrow^{+}$ ($P'$, $T'$) // add a new aligned pair
  **return**
**if** $x > 0$ **and** $y > 0$ **then**
**if** $D[x, y] = D[x - 1, y - 1]$ **and** $P[x - 1] = T[y - 1]$ **then**
    ALIGN($P$, $T$, $D$, $x - 1$, $y - 1$, $P[x - 1] + P'$, $T[y - 1] + T'$)
**if** $D[x, y] = D[x - 1, y - 1] + 1$ **then**
    ALIGN($P$, $T$, $D$, $x - 1$, $y - 1$, $P[x - 1] + P'$, $T[y - 1] + T'$)
**if** $x > 0$ **and** $D[x, y] = D[x - 1, y] + 1$ **then**
    ALIGN($P$, $T$, $D$, $x - 1$, $y$, $P[x - 1] + P'$, "-" + $T'$)
**if** $y > 0$ **and** $D[x, y] = D[x, y - 1] + 1$ **then**
    ALIGN($P$, $T$, $D$, $x$, $y - 1$, "-" + $P'$, $T[y - 1] + T'$)
```

FIGURE 3.3 Algorithm for computing the optimal alignments of *P* and *T*. Adapted from MacKenzie and Soukoreff (2002b) and Wobbrock and Myers (2006d).

3.3.2 Uncorrected Errors in Transcribed Strings

Recall the three MSD error-correction operations: *insertion, omission,* and *substitution* (see Section 3.2.2). Although the MSD algorithm tells us how many of these corrections (and hence, errors) separate two strings, it does not tell us *which* errors exist or for which characters. For that, we need a character-level error analysis for transcribed strings (MacKenzie & Soukoreff, 2002b; Wobbrock & Myers, 2006d). Consider our earlier example:

```
P: quickly
T: qucehkly
```

Previously, we determined that the MSD = 3. What might these three errors be? We could have a substitution of "c" for "i", another of "e" for "c", and an insertion of "h". Or we could have an omission of "i" and insertions of "e" and "h". Might there be other sets of three errors? How can we know?

The answer lies in using the *D* matrix filled in by the MSD algorithm (see Section 3.2.2) to produce pairs of *P* and *T* that reflect the different error corrections that result in the MSD value. These pairs are called the *optimal alignments* of *P* and *T*, and they can be discovered using the algorithm shown in Fig. 3.3.

The algorithm in Fig. 3.3 takes as parameters the presented string *P*, transcribed string *T*, the filled MSD matrix *D* (see Fig. 3.1), integers *x* and *y* (initialized with the lengths of *P* and *T*, respectively), and strings P' and T', initially empty. The last parameter is a reference to the set of optimal alignments, which is initially empty but serves as the return value once filled.

Using the ALIGN algorithm, we receive the following optimal alignment pairs of "quickly" and "qucehkly":

```
P1: qu-ickly
T1: qucehkly
```

```
P2: qui-ckly
T2: qucehkly
P3: quic-kly
T3: qucehkly
P4: quic--kly
T4: qu-cehkly
```

In the alignment pairs, hyphens in *P* indicate insertions in *T*; hyphens in *T* indicate omissions from *P*; and different letters in corresponding positions indicate substitutions. The alignments show us that there are four sets of three errors that result in MSD = 3. Since it is impossible to know which set reflects the user's actual intentions, we can simply weight each error by the number of alignments in which it occurs. For example, two alignments show a substitution of "c" for "i"; we can therefore treat this as a 50% chance that "c" was substituted for "i". Accordingly, we can tally character-level errors for each character in our alphabet. This information can help designers target problematic characters and fix them. A problematic character may be a unistroke gesture that is commonly misrecognized or confused with another, or mini-Qwerty keys that are too close together and often accidentally interchanged.

3.3.3 Corrected Errors in Input Streams

The character-level error analysis in the previous section gives fine-grained results for each letter of the alphabet, but it has a major drawback: it considers only presented and transcribed strings, ignoring the text entry process captured by input streams.

Recent work has attempted to extend the character-level error analysis to the input stream by dealing with *corrected* character-level errors, not just uncorrected errors (Wobbrock & Myers, 2006d). In many text entry studies, the number of corrected errors (i.e., backspaced letters) is much greater than the number of uncorrected errors left in the transcribed string. Therefore, a character-level error analysis of the input stream has access to a great deal more character-level data than analyses of just presented and transcribed strings. The algorithms for automatically detecting and classifying character-level input stream errors are lengthy and complicated, and interested readers are referred to them elsewhere (Wobbrock & Myers, 2006d). The remainder of this section gives examples of corrected character-level errors for illustration.

Corrected-and-Wrong, Corrected-but-Right

We have previously noted that the IF class of backspaced characters is just a count of erroneous letters and therefore may include already-correct letters. An input stream character-level error analysis can separate corrected-and-wrong from corrected-but-right characters. Consider the following example:

```
 P: quickly
IS: pui<<<quickly
```

In this input, the "p" is corrected-and-wrong, but the first "ui" is corrected-but-right. At the character level, the "p" is classified as a corrected substitution for "q", and the "ui" is corrected-but-right.

Separating corrected-and-wrong from corrected-but-right requires the determination of the target letter in *P* that the participant is entering. This is a complicated issue that involves many ambiguities. However, in the majority of cases, we can use some fairly reliable assumptions to simplify the challenges involved. For more information in this regard, readers are referred elsewhere (Wobbrock & Myers, 2006d).

Corrected Substitutions and Confusion Matrices

As we saw in the previous example, corrected substitutions occur whenever a character that does not match the target character is initially entered but later erased. Consider this example:

```
 P: quickly
IS: qv<w<vickly
```

This input shows trouble entering the "u" in "quickly". The participant first entered a "v", then erased it only to enter a "w", and then erased *that* only to enter a "v" again. Presumably the participant gave up, choosing to leave the final "v" and move on. Here, the first "v" and the "w" are corrected substitutions, while the last "v" is an uncorrected substitution.

Both uncorrected and corrected substitutions can be graphed in a *confusion matrix* (Grudin, 1984; MacKenzie & Soukoreff, 2002b; Wobbrock & Myers, 2006d). In a confusion matrix, one axis holds target characters, another axis holds produced characters, and a third axis holds the count of such entries. Figure 3.4 shows an example from a real text entry study (Wobbrock *et al.*, 2003a):

Corrected Insertions

Sometimes characters are initially inserted but then removed. In such cases, we have corrected insertions:

```
 P: quickly
IS: qxu<<uickly
```

In this example, it seems likely that the "x" was inserted before the "u", and then noticed and corrected. The "x" is therefore a corrected insertion, and the first "u" is corrected-but-right.

Corrected Omissions

A character that is initially omitted but later entered is a corrected omission. Consider:

```
 P: quickly
IS: qic<<uickly
```

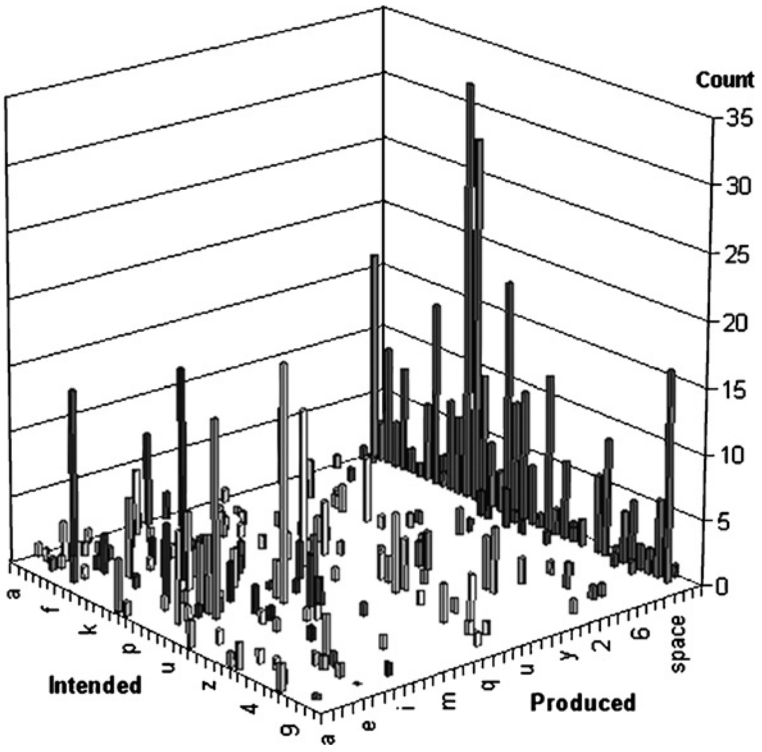

FIGURE 3.4 A confusion matrix showing various counts. The back wall shows nonrecognitions that occurred for different attempted characters. Adapted from Wobbrock and Myers (2006d). Used with permission.

Here, it seems that the “u” was initially omitted, but later replaced after backspacing the “ic”. Thus, “u” is a corrected omission, and the “ic” are corrected-but-right.

Although these corrected character-level errors seem straightforward, numerous nuances and tricky cases exist. Interested readers are referred to the fuller discussion available elsewhere (Wobbrock & Myers, 2006d).

3.4 MEASUREMENTS FROM LOG FILES

This chapter has thus far been primarily concerned with text entry experiments in which participants transcribe a presented string. Such laboratory studies are extremely useful for eliciting results in controlled settings. But what about real-world use? How can we assess a text entry method as it is used in the field? What if a particular method requires substantial learning beyond that which is possible or desirable in a single session or even multisession study? When faced with these questions, some text entry researchers have used extensive log file analysis.

Real-world text entry logs are messy. They may contain thousands or even millions of character entries, thoughtful pauses, long delays, and sudden bursts of activity and numerous nonprintable character codes (e.g., cursor arrow keys). Accordingly, the types of measurements available to researchers from real-world log files usually involve comparisons of counts or ratios.

One extended analysis logged all desktop keystrokes for four PC users over a period of 2 months (MacKenzie & Soukoreff, 2002a). The log file analysis counted each type of keystroke and reported relative percentages for each key. Interesting findings included the affirmation that SPACE is indeed the most common key (11.29%), that BACKSPACE is second most common (7.10%), and SHIFT, DOWN ARROW, and DELETE figure prominently in the results.

Another log file study of extended use focused on using a trackball as a writing device and keyboard replacement (Wobbrock & Myers, 2006c). As part of a trackball text entry method called *Trackball EdgeWrite,* users can complete entire words with simple extensions to their trackball unistrokes. However, the positions of the word completions take time to learn. Thus, extended logging of one participant with a spinal cord injury over 11 weeks was used to capture real-world learning. The results showed that 43.9% of the participant's text was entered using word completion and that undone completions accounted for only 7.7% of all completions performed, suggesting reasonable accuracy.

Of course, many more measures are possible from the analysis of real-world text entry logs. Which measures are most important will depend on the methods under investigation and the questions being asked by researchers or developers. Of course, permission should be obtained from participants before logging their text!

3.5 METHOD-SPECIFIC MEASURES

This chapter has been devoted mainly to method-agnostic measures so as to remain relevant to as many text entry methods as possible. However, some useful measures are specific to certain types of text entry methods. Those measures are covered briefly in this section.

3.5.1 Keyboard Typing

Typing on a physical keyboard has been studied extensively (Rumelhart & Norman, 1982; Grudin, 1984). Although substitutions, insertions, and omissions characterize a large percentage of typing errors (Gentner *et al.*, 1984), other errors also appear frequently in typing. For example, a *transposition error* occurs when adjacent characters are swapped:

```
P: special
T: speical
```

In the example above, the "ci" in "special" have been swapped as "ic", constituting a transposition error. In one study, researchers observed about one transposition error for every 1800 typed characters (Rumelhart & Norman, 1982). Reportedly, a large percentage of transposition errors, about 76%, occur across hands.

Another typing error is a *doubling error,* which occurs when a word that contains a double letter has the wrong letter doubled. An example is:

```
P: school
T: scholl
```

Alternation errors are similar to doubling errors but with an alternating sequence of characters. For example:

```
P: these
T: thses
```

A *homologous error* occurs when the wrong hand is used with the otherwise correct finger and key position. This type of error is thought to be more common in novice typists than in expert typists.

A *capture error* occurs "when one intends to type one sequence, but gets 'captured' by another that has a similar beginning" (Norman, 1981; Rumelhart & Norman, 1982). Capture errors occur at the word level, as in the following examples from Rumelhart and Norman (1982):

```
P: efficiency
T: efficient
P: incredibly
T: incredible
P: normal
T: norman
```

Another word-level error is a *phonetic swap,* such as in the following examples:

```
P: their
T: there
P: your
T: you're
```

These are high-level errors that are unlikely to occur in short transcription typing, but may occur when participants are composing text or transcribing long documents. Most studies that analyze these types of errors simply report the number or rate of occurrences and give salient examples.

3.5.2 Selection-Based Methods

Selection-based methods are those that show on-screen options, such as letters in a virtual keyboard, and allow users to select from among them. Selections may be performed *directly,* such as when tapping with a stylus or clicking with a mouse, or *indirectly,* such as when moving a selector with directional arrow keys and entering the highlighted letter by pressing a separate select button.

Deviation in Direct Selection

For *direct* selection-based methods, we can analyze the distribution of selection points relative to target centers and target sizes. Then we can compare deviations. A deviation value of 0% means that the selection points each hit their targets at the ideal locations (e.g., the very centers of virtual keyboard keys). A value of 100% means that the selection points were at their full allowable range (e.g., the borders of virtual keyboard keys):

$$\text{deviation} = \frac{\displaystyle\sum_{t\in T}\sum_{i=1}^{|P_t|}\frac{\sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}}{S_t}}{\displaystyle\sum_{t\in T}|P_t|}. \qquad (3.21)$$

In Eq. (3.21), t is a target in the set of all targets T and P_t is the set of selection points (x_i, y_i) intended for target t, which has its center at (x_t, y_t). The S_t term indicates the radius of target t, which for simplicity is abstracted as a circle. The range of Eq. (3.21) is 0% to 100%, assuming all selection points fall within their intended targets. If some selection points fall outside their intended targets, the upper end of the range is unbounded. Participants who are selecting targets with 90% deviation are much closer to missing than participants who are selecting at 45% deviation. An important aspect of this formula is that it is target-relative and therefore capable of comparing results obtained from different targets.

Selector Movement for Indirect Selections

Measures that can be reported for *indirect* selection-based methods are concerned with the amount and manner of selector movement. An example of an indirect selection-based method is the three-key date stamp method (MacKenzie, 2002a; Wobbrock *et al.*, 2006), in which one key moves a selector left, another moves it right, and a third selects the current letter (Fig. 3.5).

A study of three-key text entry methods found that *typamatic keystrokes,* which are those produced by automatic key repeat when a key is held down, comprise a large percentage of keystroke behavior (MacKenzie, 2002a). In one three-key method,

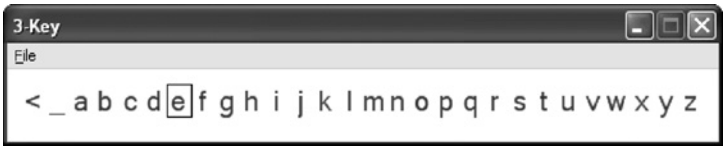

FIGURE 3.5 A three-key indirect selection-based method. The selector, which can be moved left and right, is currently over the letter "e". Image adapted from Wobbrock *et al.* (2006). Used with permission.

typamatic events comprised 77% of the observed keystrokes. In another, they comprised 42%. The selector movement was also analyzed for *movement efficiency*, since participants often overshot their intended letter. In one three-key method, participants moved the selector 13.7% more than optimal; in another, they moved it 27.7% more than necessary.

Another study examined two indirect selection-based methods used for joystick text entry on game consoles (Wobbrock *et al.*, 2004). The first was a selection keyboard. With this method, the joystick moves a selector up, down, left, and right over the keys of a virtual keyboard, and a joystick button selects the highlighted letter. Novice participants were found to move the selector about 47.6% more than necessary, in part because the keyboard employed wrap-around in both dimensions, but participants rarely used this feature. A second method was the classic in-place date stamp method familiar from high-score screens in arcade games. With this method, participants "rolled" the stamp about 21.4% more than necessary.

3.5.3 Stroke- or Gesture-Based Methods

A key trade-off when moving from selection-based methods to stroke- or gesture-based methods is in moving from *recognition* to *recall*. That is, with selection-based methods, users can recognize what actions to take based on their view of what selections are available. With strokes or gestures, however, users must remember when, how, and what actions to perform. Thus, some of the method-specific measures for gesture- based methods are often concerned with how usable, guessable, and memorable the gestures are.

Immediate Usability, Inherent Accuracy, and Memorability

MacKenzie and Zhang (1997) studied the *immediate usability* of Graffiti in four parts. First, they compared the shapes of Graffiti letters to Roman letters, finding 80.8% of them to be similar[9]. They called this value Graffiti's *inherent accuracy*. Second, they

[9] This value is not weighted by letter frequency. With letter-frequency weighting, this value is 79.2%.

exposed participants for 1 minute to a Graffiti character chart, after which participants entered each Graffiti letter five times without the chart or concern for speed. In this task, participants were 81.8% accurate. Third, they allowed participants 5 minutes of freeform practice with Graffiti, retesting them afterward as before. Accuracy increased to 95.8%. A fourth and final assessment came 1 week later when participants were brought back for a final test without additional exposure. This measure can be said to be Graffiti's *memorability* and remained at an impressive 95.8%.

Guessability

Wobbrock *et al.* (2005b) investigated the *guessability* of EdgeWrite gestures. Whereas the immediate usability procedure gave participants some initial exposure and practice, the guessability assessment did not. Twenty participants were recruited to simply guess how they would make each letter of the alphabet with their index finger on a touchpad, with the constraint that their strokes would pass through any of the four corners of the touchpad square, since this is how EdgeWrite letters are defined. Participants had no prior knowledge or exposure to EdgeWrite whatsoever.

Guessability itself was calculated by comparing participants' guesses to the existing EdgeWrite alphabet, effectively determining how many guesses would have been accommodated by the existing alphabet:

$$G = \frac{\sum_{s \in S} |P_s|}{|P|}. \tag{3.22}$$

In Eq. (3.22), G is the guessability of a symbol set S (e.g., a stroke alphabet), s is an individual symbol in S (e.g., a unistroke for "a"), P is the set of all proposals (i.e., guesses), and P_s is the set of proposals that were correct for symbol s. Using this formula, the guessability of EdgeWrite was calculated to be 51.0%. The guesses provided by participants were then collected as a *new* EdgeWrite alphabet. After resolving conflicts, this new alphabet accommodated 80.1% of participants' guesses.

3.6 DISCUSSION OF MEASURES

In this chapter, we have discussed a number of text entry measures and studies that have used them. We have reviewed aggregate measures for speed and accuracy, character-level measures, extended log file measures, and method-specific measures. However, not all measures can or should be reported for all evaluations. So when should we use which measures?

At the very least, text entry studies should report entry rate as WPM and accuracy as uncorrected errors. Neither of these essential measures should be reported without the other, since as we noted previously, they exist as trade-offs. Corrected errors are also of interest and should generally be reported, but it is important to differentiate them from uncorrected errors since corrected errors reduce WPM but uncorrected errors often increase it.

Studies of gestural methods or other methods that have clear notions of *gestures* (or *actions*) should consider reporting GPS and GPC. Studies of novices learning gestural methods should consider beginning with an assessment of guessability or immediate usability. These studies may also want to report character-level errors, particularly substitutions, and show confusion matrices in which peaks reveal commonly misrecognized or humanly confused gestures. Any other methods in which the process of making each letter differs substantially may want to report character-level errors.

Longitudinal studies should always report learning curves of WPM over sessions. They should pay particular attention to any crossover points that may occur.

Keystroke-based methods should report both performance KSPC and characteristic KSPC and compare them. They may also want to report intercharacter time, typamatic events, and typing-specific errors. Of course, other nonkeypad methods can report the KSPC performance measure as well.

Some studies will require extended learning and real-world use. In such studies, extended log file analyses will be appropriate, showing counts and ratios for different text entry events over a period of weeks or months.

Certainly there are more measures conceivable than could fit in this chapter. Different measures may be more appropriate for East Asian text entry, for example. Researchers and developers should feel encouraged to invent new measures that are targeted specifically to their method of interest. As with all measures, the important thing is that they are relevant, reproducible, and rigorous. When reported in the literature, they should be well defined and precise enough that readers can reuse the measures in their own work.

3.7 FURTHER READING

An excellent place to start is the line of work by William Soukoreff and I. S. MacKenzie (Soukoreff & MacKenzie, 2001, 2003, 2004). For character-level analyses, see MacKenzie and Soukoreff (2002b) and Wobbrock and Myers (2006d). For examples of extended log-file-based studies, see MacKenzie and Soukoreff (2002a), Magnuson and Hunnicutt (2002), and Wobbrock and Myers (2006c). A study that reports many of the measures described in this chapter for both selection-based and gestural methods is Wobbrock *et al.* (2004).

REFERENCES

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human–computer interaction.* Hillsdale, NJ: Erlbaum.

Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM, 7*, 171–176.

Evreinova, T., Evreinov, G., & Raisamo, R. (2004). Four-key text entry for physically challenged people. *Adjunct Proceedings of the 8th ERCIM Workshop on User Interfaces for All (UI4ALL '04), 28–29 June 2004, Vienna.*

Gentner, D. R., Grudin, J. T., Larochelle, S., Norman, D. A., & Rumelhart, D. E. (1984). A glossary of terms including a classification of typing errors. In W. E. Cooper (Ed.), *Cognitive aspects of skilled typewriting* (pp.39–43). New York: Springer-Verlag.

Gong, J., & Tarasewich, P. (2006). A new error metric for text entry method evaluation. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06), 22–27 April 2006, Montréal* (pp.471–474). New York: ACM Press.

Grudin, J. T. (1984). Error patterns in skilled and novice transcription typing. In W. E. Cooper (Ed.), *Cognitive aspects of skilled typewriting* (pp.121–143). New York: Springer-Verlag.

Ingmarsson, M., Dinka, D., & Zhai, S. (2004). TNT—a numeric keypad based text input method. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04), 24–29 April 2004, Vienna* (pp.639–646). New York: ACM Press.

Isokoski, P., & Kaki, M. (2002). Comparison of two touchpad-based methods for numeric entry. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '02), 20–25 April 2002, Minneapolis* (pp.25–32). New York: ACM Press.

Isokoski, P., & MacKenzie, I. S. (2003). Combined model for text entry rate development. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03), 5–10 April 2003, Ft. Lauderdale, FL* (pp.752–753). New York: ACM Press.

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163*, 845–848.

Lewis, J. R. (1999). Input rates and user preference for three small-screen input methods: Standard keyboard, predictive keyboard, and handwriting. *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting, 27 September–1 October 1999, Houston* (pp.425–428). Santa Monica, CA: Human Factors and Ergonomics Society.

Lyons, K., Starner, T., Plaisted, D., Fusia, J., Lyons, A., Drew, A., & Looney, E. W. (2004). Twiddler typing: One-handed chording text entry for mobile phones. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04), 24–29 April 2004, Vienna* (pp.671–678). New York: ACM Press.

MacKenzie, I. S. (2002a). Mobile text entry using three keys. *Proceedings of the 2nd Nordic Conference on Human–Computer Interaction (NordiCHI '02),* 19–23 October 2002, Århus, Denmark (pp.27–34). New York: ACM Press.

MacKenzie, I. S. (2002b). A note on calculating text entry speed. Unpublished work. Available online at *http://www.yorku.ca/mack/RN-TextEntrySpeed.html*

MacKenzie, I. S. (2002c). KSPC (keystrokes per character) as a characteristic of text entry techniques. *Proceedings of the 4th International Symposium on Human–Computer*

Interaction with Mobile Devices and Services (Mobile HCI02), 18–20 September 2002, Pisa (pp.195–210). Berlin: Springer-Verlag.

MacKenzie, I. S., & Soukoreff, R. W. (2002a). Text entry for mobile computing: Models and methods, theory and practice. *Human–Computer Interaction, 17,* 147–198.

MacKenzie, I. S., & Soukoreff, R. W. (2002b). A character-level error analysis technique for evaluating text entry methods. *Proceedings of the 2nd Nordic Conference on Human–Computer Interaction (NordiCHI '02), 19–23 October 2002, Århus, Denmark* (pp.243–246). New York: ACM Press.

MacKenzie, I. S., & Zhang, S. X. (1997). The immediate usability of Graffiti. *Proceedings of Graphics Interface 1997, 21–23 May 1997, Kelowna, BC* (pp.129–137). Toronto: Canadian Information Processing Society.

MacKenzie, I. S., & Zhang, S. X. (1999). The design and evaluation of a high-performance soft keyboard. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99), 15–20 May 1999, Pittsburgh* (pp.25–31). New York: ACM Press.

Magnuson, T., & Hunnicutt, S. (2002). Measuring the effectiveness of word prediction: The advantage of long-term use. *Speech, Music and Hearing, 43,* 57–67.

Matias, E., MacKenzie, I. S., & Buxton, W. (1996). One-handed touch-typing on a QWERTY keyboard. *Human–Computer Interaction, 11,* 1–27.

Morgan, H. L. (1970). Spelling correction in systems programs. *Communications of the ACM, 13,* 90–94.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review, 88,* 1–15.

Perlin, K. (1998). Quikwriting: Continuous stylus-based text entry. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98), 1–4 November 1998, San Francisco* (pp.215–216). New York: ACM Press.

Rumelhart, D. E., & Norman, D. A. (1982). Simulating a skilled typist: A study of skilled cognitive–motor performance. *Cognitive Science, 6,* 1–36.

Soukoreff, R. W., & MacKenzie, I. S. (2001). Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '01), 31 March–5 April 2001, Seattle* (pp.319–320). New York: ACM Press.

Soukoreff, R. W., & MacKenzie, I. S. (2003). Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03), 5–10 April 2003, Ft. Lauderdale, FL* (pp.113–120). New York: ACM Press.

Soukoreff, R. W., & MacKenzie, I. S. (2004). Recent developments in text-entry error rate measurement. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '04), 24–29 April 2004, Vienna* (pp.1425–1428). New York: ACM Press.

Venolia, D., & Neiberg, F. (1994). T-Cube: A fast, self-disclosing pen-based alphabet. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94), 24–28 April 1994, Boston* (pp.265–270). New York: ACM Press.

Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery, 21*, 168–173.

Wobbrock, J. O., Aung, H. H., Myers, B. A., & LoPresti, E. F. (2005a). Integrated text entry from power wheelchairs. *Journal of Behaviour and Information Technology, 24*, 187–203.

Wobbrock, J. O., Aung, H. H., Rothrock, B., & Myers, B. A. (2005b). Maximizing the guessability of symbolic input. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05), 2–7 April 2005, Portland, OR* (pp.1869–1872). New York: ACM Press.

Wobbrock, J. O., & Myers, B. A. (2006a). Trackball text entry for people with motor impairments. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06), 22–27 April 2006, Montréal* (pp.479–488). New York: ACM Press.

Wobbrock, J. O., & Myers, B. A. (2006b). In-stroke word completion. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '06), 15–18 October 2006, Montreux, Switzerland* (pp.333–336). New York: ACM Press.

Wobbrock, J. O., & Myers, B. A. (2006c). From letters to words: Efficient stroke-based word completion for trackball text entry. *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '06), 23–25 October 2006, Portland, OR* (pp.2–9). New York: ACM Press.

Wobbrock, J. O., & Myers, B. A. (2006d). Analyzing the input stream for character-level errors in unconstrained text entry evaluations. *Transactions on Computer–Human Interaction (TOCHI)*, 13(4), 458–489.

Wobbrock, J. O., Myers, B. A., & Aung, H. H. (2004). Writing with a joystick: A comparison of date stamp, selection keyboard, and EdgeWrite. *Proceedings of Graphics Interface 2004, 17–19 May 2004, London, ON* (pp.1–8). Waterloo, ON: Canadian Human–Computer Communications Society.

Wobbrock, J. O., Myers, B. A., & Hudson, S. E. (2003a). Exploring edge-based input techniques for handheld text entry. *Proceedings of the 3rd International Workshop on Smart Appliances and Wearable Computing (IWSAWC '03).* In *23rd International Conference on Distributed Computing Systems Workshops (ICDCSW '03), 19–22 May 2003, Providence, RI* (pp.280–282). Los Alamitos, CA: IEEE Computer Society.

Wobbrock, J. O., Myers, B. A., & Kembel, J. A. (2003b). EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03), 2–5 November 2003, Vancouver, BC* (pp.61–70). New York: ACM Press.

Wobbrock, J. O., Myers, B. A., & Rothrock, B. (2006). Few-key text entry revisited: Mnemonic gestures on four keys. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06), 22–27 April 2006, Montréal* (pp.489–492). New York: ACM Press.

Yamada, H. (1980). A historical study of typewriters and typing methods: From the position of planning Japanese parallels. *Journal of Information Processing, 2,* 175–202.