MOBILE TEXT ENTRY USING AMBIGUOUS KEYPADS: NEW METRICS IN A NEW TOOLKIT

STEVEN JOHN CASTELLUCCI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAMME IN COMPUTER SCIENCE AND ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO

MAY 2007

MOBILE TEXT ENTRY USING AMBIGUOUS KEYPADS:
NEW METRICS IN A NEW TOOLKIT

by **Steven John Castellucci**

A thesis submitted to the Faculty of Graduate Studies of
York University in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

MOBILE TEXT ENTRY USING AMBIGUOUS KEYPADS:
NEW METRICS IN A NEW TOOLKIT

by **Steven John Castellucci**

By virtue of submitting this document electronically, the author certifies that this is a true electronic equivalent of the copy of the thesis approved by York University for the award of the degree. No alteration of the content has occurred and if there are any minor variations in formatting, they are as a result of the conversion to Adobe Acrobat format (or similar software application).

Examination Committee Members:

1. Doctor Scott MacKenzie
2. Doctor Wolfgang Stuerzlinger
3. Doctor Vassilios Tzerpos
4. Doctor Stephen Chen

ABSTRACT

Technical advancements involving the pervasive cell phone make it a viable computing platform. As such, the input of textual information is vital. Use of ambiguous keypads preserves the device's portable nature by mapping multiple letters to a single key. Metrics exist to assess such keypads, but they do not account for the physical resources employed by keypads, or for the perceptual and cognitive load placed on the user. Furthermore, existing tools typically encapsulate only one metric, making detailed evaluation cumbersome and time-consuming.

This thesis outlines a new metric that quantifies a keypad's efficiency using the number of text entry keys it employs. It also presents a revised performance model that incorporates perceptual and cognitive timing values to reflect actual practice. Encompassing both contributions is an original toolkit that simplifies and streamlines analysis of ambiguous keypads.

ACKNOWLEDGEMENTS

I would like to convey my sincerest gratitude to my supervisor, Dr. Scott MacKenzie, for his knowledge, guidance, and support. His generous funding of me via his NSERC grant made this research possible.

Thanks also go to Dr. Wolfgang Stuerzlinger, who introduced me to the graduate program, Dr. Bil Tzerpos, and Dr. Stephen Chen for taking time out of their busy schedules to serve on my thesis examination committee.

In addition, thanks go to Chris Klochek, who served as a preliminary participant for the evaluation described in Chapter 3, and to Andriy Pavlovych and Aleks Oniszczak, who helped produce a video profiling TnToolkit.

Most importantly, I would like to thank my parents, Betty and Sandro, for their love, encouragement, and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1
Introduction

Mobile devices such as personal digital assistants (PDAs) and mobile phones (a.k.a. cell phones) are immensely popular, with worldwide first quarter 2006 sales of approximately 228.2 million (Shirer, Baker, & Llamas, 2006; Shirer & Llamas, 2006). However, the mobile phone occupies an overwhelming preference, as its sales over the same timeframe dwarf those of PDAs by a ratio of 150:1 (Shirer et al., 2006; Shirer & Llamas, 2006). However, despite its name, the mobile phone's functionality is not limited to facilitating voice communication. Modern features, such as the ability to email, instant message, navigate the Internet, manage personal contacts, play music files, take pictures, capture video, and play video games, has led to the proliferation of mobile phones as both portable communication tools and portable entertainment units. With the ability to add contact information, edit song details, email, and chat, text entry on mobile phones possesses considerable importance. In addition, mobile text entry is integral to the established practice of sending Short Message Service (SMS) messages (a.k.a., sending text messages or *texting*). Mobile text entry is so significant, that, as of September 2005, an estimated 89 billion text messages were sent per month worldwide (Cellulist). Furthermore, a report cites this dextrous task as the reason for increased incidences of repetitive stress injury (RSI) in both adults and in children as young as eight years old (BBC, 2006).

As a consequence of mobile text entry's prevalence, researchers and device manufacturers are seeking techniques to improve the efficiency and performance of mobile text entry. To that end, this thesis presents new metrics to characterise text entry on mobile devices: Chapter 2 offers a quantification of a technique's efficiency, and Chapter 3 posits a performance prediction model that accounts for typical cognitive and perceptual processes. This thesis then presents TnToolkit, an innovative tool that combines established metrics with those in this thesis. Chapter 4 expounds on the features of TnToolkit, such as its ability to quickly and easily evaluate existing text entry implementations and to aid in the design of new techniques. Though each chapter concludes with a summary of work mentioned therein, Chapter 5 describes conclusions that span multiple chapter topics and proposes future research work.

The form of mobile text entry addressed in this thesis involves ambiguous keypads (a.k.a., reduced keyboards) with disambiguation technologies, as typical of most mobile devices. The remainder of this chapter introduces the ambiguous keypad and details various text entry techniques that this thesis references.

1.1 Ambiguous Keypads

Ambiguous keypads are those that assign multiple characters to a single key. Consequently, the mobile device can utilize fewer keys, allowing designers to enlarge each key without increasing the space occupied by the entire keypad (i.e., its footprint). Furthermore, compared to a mini QWERTY keypad (where each character has its own

minute key), the enlarged keys of an ambiguous keypad require less dextrous movements, and are therefore less likely to promote RSI (BBC, 2006).

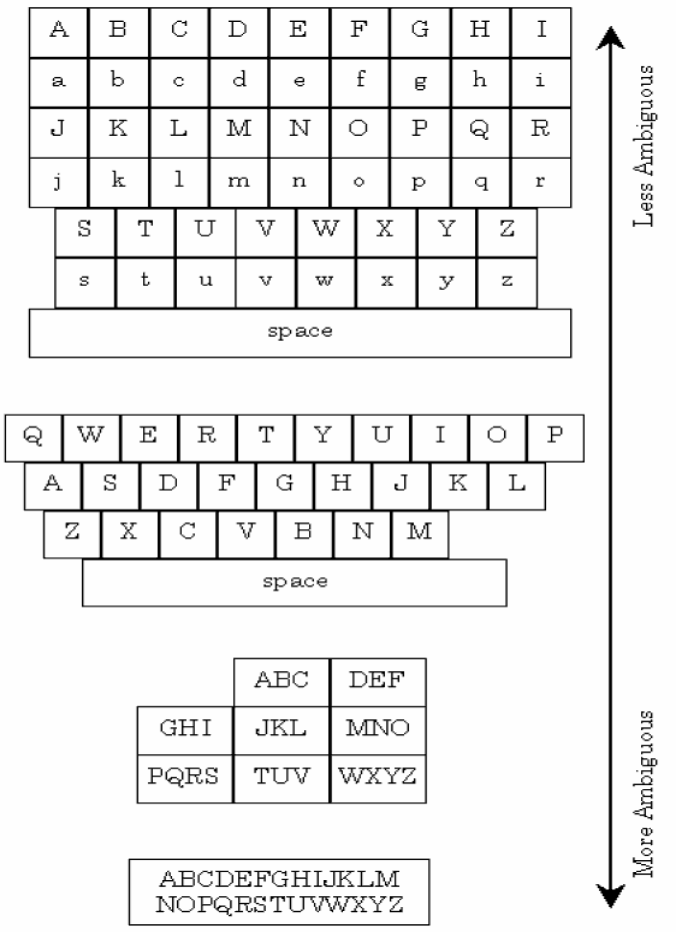

Figure 1-1: The "key-amgibuity continuum" (MacKenzie & Soukoreff, 2002b).

A paper by MacKenzie and Soukoreff (2002b) presents actual and theoretical keyboard layouts in a "key-ambiguity continuum" (Figure 1-1). At one extreme is a layout that assigns each character its own key, distinguishing between upper- and lower-case letters. At the other extreme is a purely theoretical layout with all letters mapped to

one key. Consequently, it deems the QWERTY keyboard slightly ambiguous, as a key (with use of the SHIFT modifier) can correspond to an upper- and a lower-case letter, a number and a symbol, or multiple punctuations. However, text entry models usually ignore entry of numbers and punctuations, and assume that all letters are lower-case (MacKenzie & Soukoreff, 2002b). Thus, the QWERTY keyboard is generally considered non-ambiguous.

Mapping multiple characters to a single key creates ambiguity, as a single keystroke could mean one of several characters (hence the term "ambiguous keypad"). Initially, this required users to select the desired character by tapping a key multiple times; this is known as the *Multi-tap* technique. Each key press displays the next character associated with that key. The user selects a character by pausing for a timeout period, or by pressing another key.

Contemporary techniques employ language-based disambiguation (a.k.a., dictionary-based disambiguation), wherein users press a key only once for each letter. By using a corpus of the target language, the disambiguation algorithm maps sequences of keystrokes to actual words. Occasionally, a collision occurs – a key sequence maps to multiple words. Disambiguation techniques employ a "NEXT" key to allow the user to cycle through possible words and select the intended one. To enter non-dictionary words, users can still resort to using the Multi-tap technique. Furthermore, to minimize reliance on multi-tap, users can typically augment a disambiguation technique's corpus with new words.

1.2 Text Entry Techniques

The subsequent subsections describe various text entry techniques – the keypad layout and disambiguation method. Though each technique incorporates its own proprietary (and sometimes user-expandable) corpus, this thesis applies the following (static) independent corpora to allow for consistent comparisons:

- BNC1: A corpus based on the 9022 most frequent words in the British National Corpus (BNC; Silfverberg, MacKenzie, & Korhonen, 2000).
- BNC2: A corpus based on the 64 566 most frequent words in the British National Corpus (BNC; MacKenzie, 2002).
- SMS: A corpus with 7189 distinct words representing text messaging vocabulary (How & Kan, 2005). The inclusion of this corpus aims to address the concern that standard corpora fail to reflect the actual vernacular employed by mobile users (Soukoreff & MacKenzie, 2003).

To allow for further consistency in comparisons, this thesis will not explore technique enhancements such as word completion and word prediction. Such features are not available with all implementations and unnecessarily complicate text entry models. For example, the corpus changes to reflect a user's written vocabulary and the input required for a word becomes dependent on those previously entered.

There exist numerous ambiguous keypads in academic literature and commercial devices. However, the following ones are discussed throughout this thesis:

1.2.1 T9

Developed by Tegic Communications (www.tegic.com), *T9* is available on more than 800 million mobile handsets worldwide and in more than 50 languages (T9). *T9* (which stands for "Text on 9 keys" (T9)) uses the ubiquitous standard telephone keypad (Figure 1-2) (officially known as ITU E.161 (Hissen)) for text entry. When collisions occur, *T9* lists all possible words (i.e., all words that map to the entered key sequence) in descending order of frequency in the corpus. For example, the key sequence *2-2-5-3* could represent the word *cake*, *able*, *bald* or *calf*. Consequently, additional input is required to identify the desired one. Entering "able", affixed with the obligatory space character, would involve one keystroke for each letter, one press of the NEXT-key to select "able" from the list, and one press of the SPACE-key to terminate the word's entry. The NEXT-key sometimes differs between implementations, but is typically the *-key.

Figure 1-2: An example of the standard mobile telephone keypad.

1.2.2 SureType

The *SureType* keypad (Figure 1-3) (BlackBerry) maps the traditional QWERTY layout to 15 keys by assigning pairs (typically) of adjacent letters to the same key. Like *T9*, words involved in a collision appear in descending order of frequency in the corpus. To select the intended word, the user may use the device's scroll wheel or the *-key as the NEXT-key.

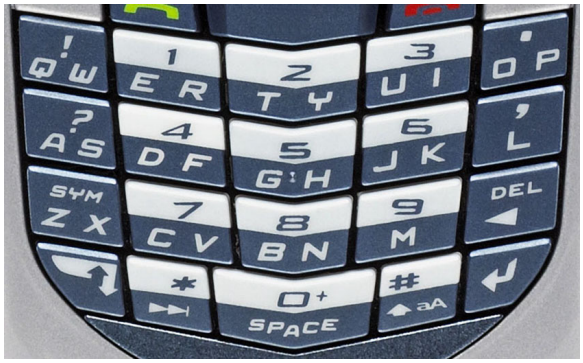

Figure 1-3: The SureType keypad (www.blackberry.com).

1.2.3 EQx

EQx (Eatoni) (which stands for "Eatoni QWERTY with *x* columns") attempts to map the QWERTY layout to the available keys on a device while minimizing the probability of collisions. For example, while the letters "L" and "U" are not adjacent on QWERTY keyboards, they are mapped to the same key in the *EQ3* (Figure 1-4) and *EQ6* (Figure 1-5) layouts. Consequently, the layout is not strictly QWERTY, but rather QWERTY-like.

Available in 165 languages, *EQx* operates in two modes: *WordWise* and *LetterWise* (Eatoni). Like *T9* and *SureType*, *WordWise* is a dictionary-based disambiguation technique. However, to enter a word not in the corpus, *EQx* uses

LetterWise instead of Multi-tap. With a single key press, *LetterWise* predicts the intended character based on those previously entered and the statistical distribution of characters within the target language (MacKenzie, Kober, Smith, Jones, & Skepner, 2001). While usually accurate, the user can press the NEXT-key to enter the next likely character mapped to that key.

Figure 1-4: An implementation of EQ3 on a Nokia 6680.

Figure 1-5: An example of the EQ6 keypad layout.

Chapter 2
Quantifying Ambiguous Keypad Efficiency

Mobile devices that provide key-based input are ubiquitous. However, their design poses a dilemma: large keypads provide desktop practicality, but portable devices favour a small form factor. By mapping multiple letters to a single key, ambiguous keypads provide a convenient balance. (In addition, their reduced size also facilitates text entry by users with impaired motor skills (Koester & Levine, 1994; Kuhn & Garbe, 2001; Lesher, Moulton, & Higginbotham, 1998).) However, despite their proliferation, there exists no measurement to quantify a technique's approach to combining functionality and compactness.

This chapter first presents a new metric that combines a technique's keystrokes per character (KSPC) measurement with the number of keys it employs. The result is a key-character mapping efficiency (KCME) value that developers can calculate in the development stages of a new technique, and is representative of the average keystrokes and physical resources (i.e., keys) it requires. This chapter then evaluates and compares the KCME of several text entry techniques.

2.1 Quantifying Efficiency

Mobile devices employ a variety of text entry methods. Traditional BlackBerry devices use a miniature QWERTY keypad, and though this technique requires at least 27 keys for text entry (one for each letter of the English alphabet and one for the SPACE character), it

allows users to enter characters with a single keystroke. At the opposite extreme, some devices (e.g. early-model pagers) use only a few keys to select characters from a list. However, the consequence of so few keys is that one typically requires many keystrokes to enter a character. Ambiguous keypads allow for the use of fewer than 27 keys for text entry, though it typically requires a few supplementary keystrokes to disambiguate input.

A characteristic of text entry techniques is the keystrokes-per-character (KSPC) measurement. Using a corpus of the target language, it describes the average number of keystrokes required by the user to input a character in a given language using a given interaction technique (MacKenzie, 2002). However, KSPC does not factor-in the spatial requirement (i.e., the number of keys used) for a text input technique, which is significant in the design of mobile devices. For example, take the following two mobile text entry techniques:

Technique A: 20-key keypad with a KSPC of 1.0001
Technique B: 10-key keypad with a KSPC of 1.0064

Based on KSPC alone, *Technique A* appears more favourable. However, when also considering keypad size, *Technique B*'s KSPC value is only 0.63% greater and uses half the number of keys. This example clearly illustrates the value of integrating both a technique's KSPC and its number of keys into a quantitative metric. By using the number of keys rather than the actual space occupied by the keypad, calculations employ technique-specific details, which are static, rather than implementation details, which often vary between device models. Consequently, this maintains the simplicity of the metric.

Equation 2-1 represents this author's metric for quantifying the key-character mapping efficiency (KCME) of a mobile text entry technique. While (with respect to engineering) efficiency can represent the ratio of work performed to energy expended, this metric represents efficiency using the average keystrokes and the physical resources (i.e., keys) required by a specific mobile text entry technique. It combines a technique's KSPC27 value (calculated in this thesis using BNC1) with the number of keys it utilizes and expresses efficiency in relation to the omnipresent QWERTY technique. The variable *numKeys* represents the number of keys employed by a technique for text entry (i.e., those associated with letters, the SPACE-key, and the NEXT-key). Although similar to *T-factor* (MacKenzie & Tanaka-Ishii, in press), this value also includes the NEXT-key, and thus represents *T-factor*+1.

$$KCME_A = \frac{numKeys_{QWERTY} * KSPC27_{QWERTY}}{numKeys_A * KSPC27_A} = \frac{27 * 1}{numKeys_A * KSPC27_A}$$

Equation 2-1: The efficiency metric for Technique A*, relative to QWERTY.*

$$KCME_{T9} = \frac{27 * 1}{numKeys_{T9} * KSPC27_{T9}} = \frac{27}{10 * 1.0064} = 2.6828$$

Equation 2-2: Calculating KCME for T9.

$$KCME_{SureType} = \frac{27 * 1}{numKeys_{SureType} * KSPC27_{SureType}}$$
$$= \frac{27}{16 * 1.0020}$$
$$= 1.6842$$

Equation 2-3: Calculating KCME for SureType.

$$KCME_{EQ3} = \frac{27 * 1}{numKeys_{EQ3} * KSPC27_{EQ3}}$$
$$= \frac{27}{12 * 1.0023}$$
$$= 2.2449$$

Equation 2-4: Calculating KCME for EQ3.

$$KCME_{EQ6} = \frac{27 * 1}{numKeys_{EQ6} * KSPC27_{EQ6}}$$
$$= \frac{27}{20 * 1.0001}$$
$$= 1.3499$$

Equation 2-5: Calculating KCME for EQ6.

KCME values greater than 1 represent superior efficiency, while values less than 1 represent inferior efficiency with respect to the QWERTY technique. However, this need not be the case. Equation 2-6 reveals how efficiency can be calculated using a different technique as the benchmark.

$$KCME_{(A,B)} = \frac{\left(\frac{numKeys_{QWERTY} * KSPC27_{QWERTY}}{numKeys_A * KSPC27_A} \right)}{\left(\frac{numKeys_{QWERTY} * KSPC27_{QWERTY}}{numKeys_B * KSPC27_B} \right)} = \frac{numKeys_B * KSPC27_B}{numKeys_A * KSPC27_A}$$

Equation 2-6: The efficiency metric for Technique A*, relative to* Technique B.

2.2 Comparison of Ambiguous Text Entry Techniques

Figure 2-1 illustrates the significance of KCME values. Techniques that map to locations on the blue power curve are just as efficient as QWERTY and thereby represent KCME values of one. Techniques that map to locations below the curve are more efficient than QWERTY and have values greater than one, while those above the curve are less efficient than QWERTY and have values less than one. Theoretically, the closer to the origin a technique maps, the more efficient it is.

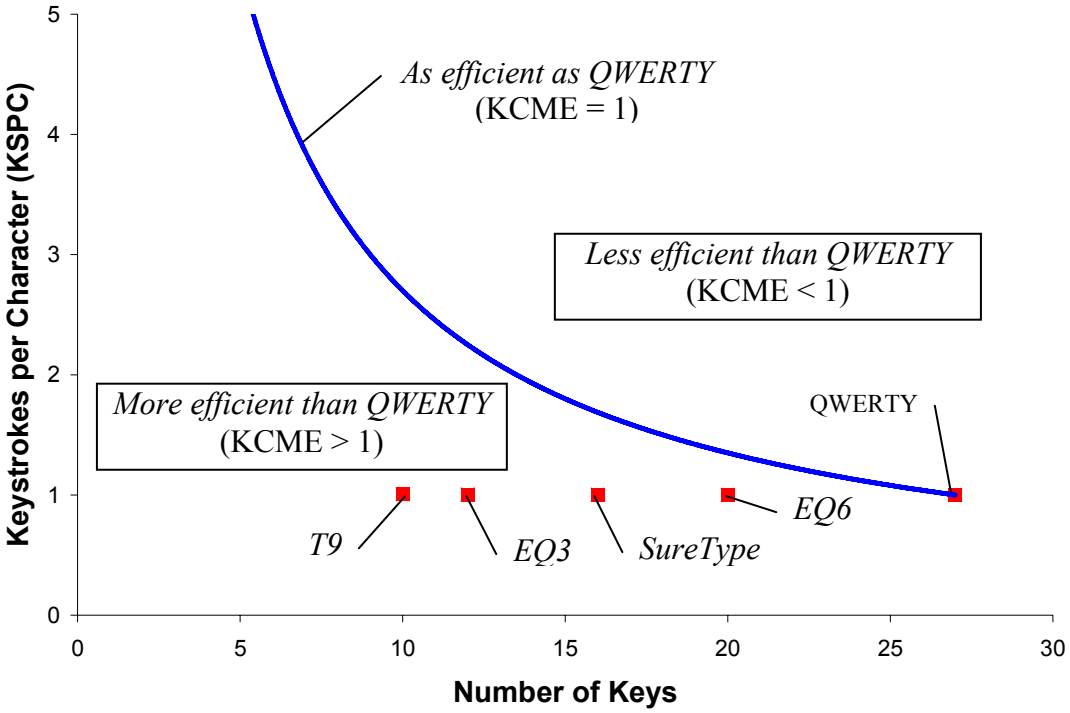

Figure 2-1: This graph illustrates the significance of KCME values.

In addition to the aforementioned input techniques, there exist additional ambiguous keypad designs in academia (Figure 2-2 to Figure 2-9 inclusive). While some strive to reduce the number of keystrokes required on a 12-key (i.e., telephone) keypad, others facilitate text entry using even fewer keys.

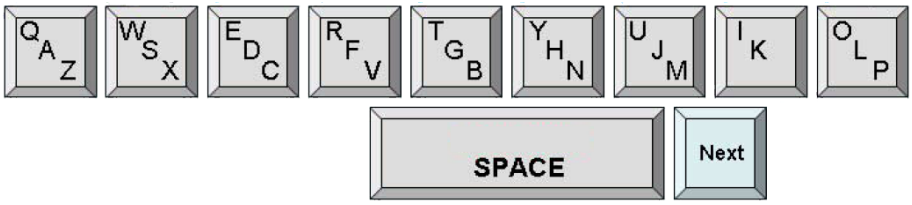

Figure 2-2: The Stick *(Green, Kruger, Faldu, & Amant, 2004) keypad.*

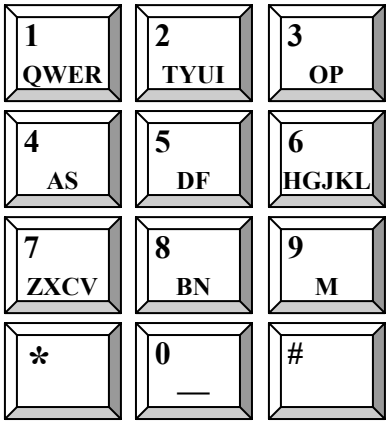

Figure 2-3: The QP10213 *(MacKenzie & Tanaka-Ishii, in press) keypad.*

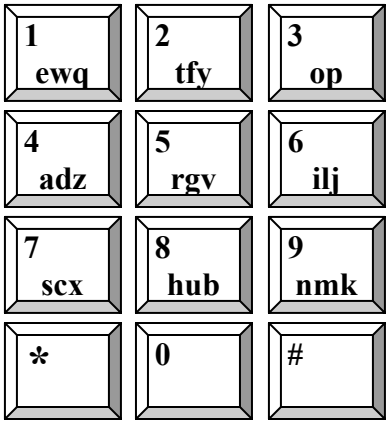

Figure 2-4: The Qwerty-Like Phone Keypad (QLPK) (Hwang & Lee, 2005).

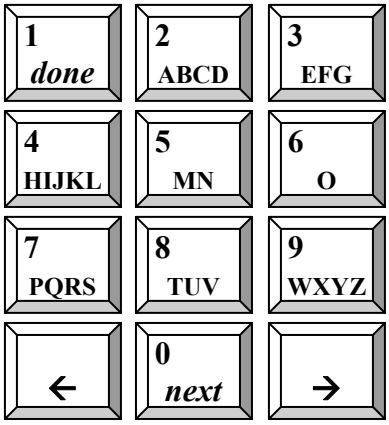

Figure 2-5: The Alphabetically Constrained Design (ACD) (Gong & Tarasewich, 2005) keypad.

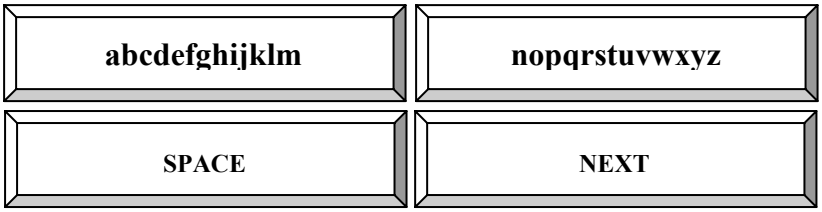

Figure 2-6: The Letters on 2 Keys (L2K) *(MacKenzie & Tanaka-Ishii, in press) keypad.*

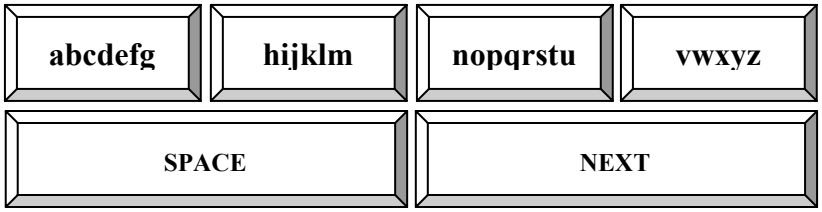

Figure 2-7: The Letters on 4 Keys (L4K) *(MacKenzie & Tanaka-Ishii, in press) keypad.*

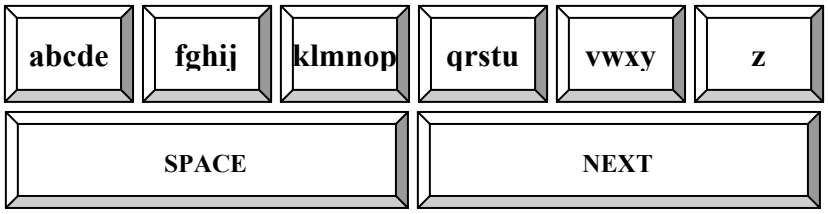

Figure 2-8: The Letters on 6 Keys (L6K) *(MacKenzie & Tanaka-Ishii, in press) keypad.*

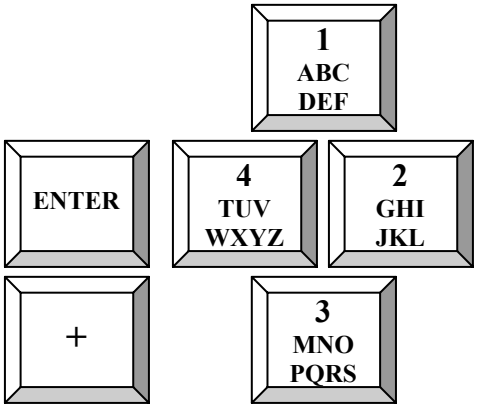

Figure 2-9: The TouchMeKey4 (TMK4) *(Tanaka-Ishii, Inutsuka, & Takeichi, 2002) keypad.*

Table 2-1 tabulates the KCME values of four mobile text entry techniques. By mapping only a few letters (and sometimes only one) to a key, *SureType*, *EQ3*, and *EQ6* yield KSPC values very close to one. However, while these technologies achieve the best KSPC measurements, accounting for the large number of keys required to achieve such results (16, 12, and 20 keys, respectively) reveals KCME values that are not the best. Instead, the most efficient techniques evaluated are *T9* (commercial) and *L2K* (academic). While they do not possess the best KSPC measurement amongst the group (*L2K* has the worst), their limited use of physical resources (in the form of only 10 and 4 keys, respectively) benefits their efficiency.

Technique	Number of Keys	KSPC27 Value	KCME Value
L2K	4	1.5471	4.3629
TMK4	6	1.0512	4.2809
L4K	6	1.0670	4.2176
L6K	8	1.0288	3.2804
ACD	10	1.0058	2.6846
T9	10	1.0064	2.6828
QP10213	11	1.0043	2.4440
QLPK	11	1.0054	2.4414
Stick	11	1.0055	2.4412
EQ3	12	1.0023	2.2449
SureType	16	1.0020	1.6842
EQ6	20	1.0001	1.3499

Table 2-1: Characteristics and KCME values of mobile text entry methods.

2.3 Summary

With the popular use of mobile devices for text messaging, instant messaging, and emailing, efficient methods of text entry is essential. Furthermore, while metrics exist to evaluate text entry throughput, none exists to appraise a technique's balance of functionality and compactness. This chapter presented KCME, a new metric for assessing such efficiency, which combines a technique's KSPC measurement with the number of keys it employs. Using this metric, *T9* is the most efficient commercial input method with a KCME value of 2.68, and *L2K* is the most efficient academic design with a value of 4.36. Though its KSPC measurement is the highest amongst evaluated techniques, *L2K* uses only 4 keys for text entry.

By using only the key-character mapping and the number of keys required by a text entry technique, evaluators can calculate an efficiency measurement during the design phase, without the need for implementation details.

Chapter 3
A Revised Performance Model for Ambiguous Input

Previous research has yielded models to predict upper-bound text entry performance in words per minute (wpm) (MacKenzie & Soukoreff, 2002a; Silfverberg et al., 2000). In addition, while metrics gathered from such models can benefit the design of improved techniques, some believe that enhancing a technique's ease-of-use (and, consequently, its maximum achievable performance) can actually benefit the user's writing style (Yamada, 1980).

However, research illustrates the disparity between predicted and actual performance (James & Reischel, 2001). Furthermore, despite the popular use of ambiguous keyboards for mobile text entry, no previous model specifically accounts for the mental overhead incurred by the associated perceptual and cognitive processes. The revised model, presented as the topic of this chapter, increases the time to enter each word in the corpus to account for the performance cost of perceptual and cognitive loads on the user.

Research by Pavlovych and Stuerzlinger (2004) takes a similar approach, but with significant differences. Firstly, their operators and timing values differ from those employed by the revised model. While their timing values include empirically determined visual scan time, they apply specifically to one-thumb use of a 12-key telephone keypad.

Application of their model to other layouts would require additional empirical investigation to determine its applicable timing values. Furthermore, while their methodology models novice use, the revised model predicts the upper-bound text entry performance with various levels of task familiarity. In addition, their model concentrates on characters entered. While this makes it adaptable for various forms of text entry, this author's revised model focuses on inputted words, and thus, is more suitable for the conventional ambiguous text entry techniques presented in this thesis.

The first section of this chapter introduces concepts related to this research. Subsequent sections present the revised model in detail, evaluate it, and reveal the evaluation results.

3.1 Topic Primer

This section gives brief introductions to three integral concepts related to this research: Fitts' law, the Model Human Processor (MHP), and the Keystroke-Level Model (KLM).

3.1.1 Fitts' Law

Prevalent in the field of Human Factors and Human-Computer Interaction, Fitts' law (Fitts, 1954) predicts the movement time (*MT*) between two targets, each of width *W*, at a distance (a.k.a. amplitude) *A*. Its principle component is a quantification of the movement task's index of difficulty (*ID*). (Values of *ID* have the unit "bits", but do not relate to binary digits.) Though subtle variations exist in equations to calculate *ID* (MacKenzie,

1992; MacKenzie, 2003), this research employs the following version, introduced by MacKenzie (1989):

$$ID = \log_2\left(\frac{A}{W} + 1\right)$$

Equation 3-1: The equation for ID.

The Fitts' law equation for movement time is the following:

$$MT = a + b \times ID$$

Equation 3-2: The Fitts' law equation for MT.

Empirical tests (typically using linear regression (MacKenzie, 2003)) yield the intercept and slope coefficients of this linear equation. If they have units of milliseconds (ms) and ms/bit, respectively, then the prediction of *MT* is in ms.

3.1.2 Model Human Processor

Proposed by Card, Moran, and Newell, the Model Human Processor (MHP) (1983) models human mental processes. These include perceiving a stimulus, and accessing short-term memory (STM) and long-term memory (LTM). For each process, it defines a typical duration (a.k.a. timing value), and a range that represents task and participant variations. The revised model associates the maximum value in each range with novice behaviour, characterised by slower responses to stimuli and hesitancy with the task (with respect to typical behaviour). Conversely, the minimum value in each range is associated

with expert behaviour, epitomizing quick responses and task proficiency. Table 3-1 tabulates relevant operators and their associated timing values.

Operator	Description	Minimum (ms)	Typical (ms)	Maximum (ms)
t_E	The time occupied with an eye movement	70	230	700
t_P	The delay between the onset of a stimulus and perception of that stimulus	50	100	200
t_M	The time to send a motor impulse	30	70	100
t_C	The time to use the contents of STM as input for an operation (e.g., accessing LTM) that change the contents of short-term memory	25	70	170
t_S	The time to determine if two words are the same	36	47	52

Table 3-1: MHP operators and their associated timing values.

3.1.3 Keystroke-Level Model

Also by Card, Moran, and Newell, the Keystroke-Level Model (KLM) (1980) defines operators to model computer input via the keyboard and mouse.

Operator	Description
K	Performing a keystroke or pressing a button
P	Pointing to a displayed target via a mouse
H	Placing or returning one's hand(s) to a rest position on the input device
D	Drawing straight-line segments
M	Mentally preparing for executing physical actions
R	Waiting for a response from the system

Table 3-2: KLM operators.

The revised model reasonably assumes entry via only a keypad and that the user's hands are on it prior to input, thus eliminating the pointing, homing, and drawing

operators. It also assumes negligible system reaction time, thus eliminating the response operator.

3.2 Perceptual and Cognitive Model

This section outlines applicable perceptual and cognitive loads associated with text entry using ambiguous keyboard. It then presents the revised performance prediction model. Throughout this section, "*W*" represents an example word. The symbol "|*W*|" represents its length and the numbers 0..|*W*|-1 represent the indices of its letters.

3.2.1 Applicable Perceptual and Cognitive Loads

Ambiguous text entry requires the user to expend perceptual and cognitive effort. Visual perception is essential for verification of a displayed word. Cognitive processes include mental preparation, storing the perceptual input, comparing the displayed word to the intended one and determining subsequent action. In general, determining the next action can involve complex problem solving. However, for text entry in particular, actions are associated with specific conditions (e.g., pressing the SPACE key at the end of a word to complete entry). The user must decide whether a particular condition holds, and if so, take the appropriate action. An interesting component of mental preparation involves spelling. The revised model reasonably assumes the user mentally composes the text to enter beforehand and that it exists in STM. However, the contents of STM are commonly acoustic in nature (Card et al., 1983) and so the user must fetch the spelling of each word from LTM.

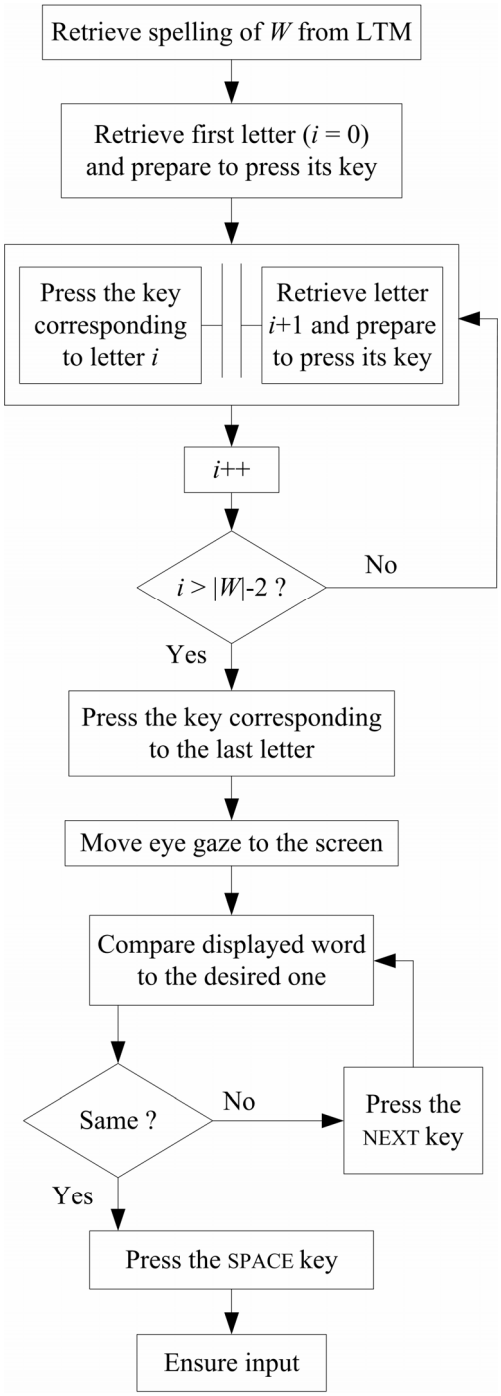

Figure 3-1: A diagram depicting the processes and decisions involved with text entry using an ambiguous keypad.

Figure 3-1 depicts the processes and decisions involved with text entry using an ambiguous keypad. The user first mentally prepares by retrieving the spelling of the word, *W*, from LTM, noting its first letter, and prepares to take appropriate action. While the user takes action to enter this letter, he or she can mentally prepare for the next letter in the word. This cycle continues until the user finishes entering the second-last letter (indexed $|W|$-2) and is prepared to enter the last one (indexed $|W|$-1). The user then enters the last letter and transfers eye gaze to the screen. This model assumes that the keypad and screen are close enough that this movement involves only a saccade and not an associated head movement. The user then compares the displayed word to the intended one and presses the NEXT-key if necessary. Pressing the SPACE-key accepts the displayed word and terminates its entry. The user can then verify input and continue with the next word.

3.2.2 Mathematical Model

Parameters inherited from traditional text entry models (Silfverberg et al., 2000) (MacKenzie & Soukoreff, 2002a) include Fitts' law coefficients and the time between successive key presses involving alternate thumbs (t_{MIN}) (MacKenzie & Soukoreff, 2002a). Unless otherwise noted, performance predictions in this thesis employ the following parameter values determined by Silfverberg et al. and MacKenzie and Soukoreff: a (intercept) = 176 ms, b (slope) = 64 ms/bit (Silfverberg et al., 2000), and t_{MIN} = 88 ms (MacKenzie & Soukoreff, 2002a). By incorporating mental timing values into the equation for movement time, the revised model simulates this additional load and

yields theoretical upper-bound performance predictions that more closely mimic actual usage. The revised model derives the necessary components from the MHP. Also of paramount importance are the KLM operators of keystroking and mental preparation. Keystroking relies on Fitts' law and mental preparation relies on time to access short-term memory (STM) and long-term memory (LTM).

Card, Moran and Newell present their timing values as a typical value within a range. The revised model associates minimum and maximum range values with novice and expert performance, respectively. However, these terms do not necessarily describe the user's familiarity with text entry in general, but rather the user's familiarity with a specific entry task. With repetition, the time spent on perceptual and cognitive aspects decreases as actions become rehearsed, fluid, and natural. This model assumes the absence of typographical errors and zero visual scan time (i.e., the user is familiar with the employed letter-key mapping and need not search for the key corresponding to the desired letter). Researchers had thought that the Hick-Hyman law (Hick, 1952; Hyman, 1953) for choice reaction time could reliably account for actual visual scan time (Soukoreff & MacKenzie, 1995), but that was later refuted (MacKenzie & Zhang, 2001).

The revised model appears in mathematical form as Equation 3-3. For simplicity, it presents the previously mentioned timings as compound variables. Table 3-3 describes the variables and their composition.

$$t_W = M + A_0 + I_0 + \left(\sum_{i=0}^{|W|-2} \max(K_i, A_{i+1} + I_{i+1}) \right) + K_{|W|-1} + E + C + n(K_N + C) + K_S + C$$

Equation 3-3: The revised model in mathematical form.

Variable	Description
t_W	The time to input a word W, indexed 0 .. $\|W\|$-1
M	The time to retrieve the spelling of W from LTM; equals t_C
A_i	The time to determine the i^{th} letter of W; equals t_C
I_i	The time to send a motor impulse to key-in the i^{th} letter of W; equals t_M
E	The time occupied with an eye movement; equals t_E
C	The time to perceive and store the displayed word in STM ($t_P + t_C$), compare it to the desired word (t_S), and initiate appropriate action ($t_C + t_M$); equals ($t_P + 2t_C + t_S + t_M$) in total
K_i	The time to key-in the i^{th} letter via Fitts' law
n	The number of required presses of the NEXT key
K_N	The time to press the NEXT key via Fitts' law
K_S	The time to press the SPACE key via Fitts' law

Table 3-3: Description of variables composing the revised model.

Regarding the summation within the large parentheses of Equation 3-3, note that the user can perform ($A_{i+1} + I_{i+1}$) in parallel with K_i for i = 0 .. $|W|$-2 (i.e., the user can prepare for the next letter while keying the current one). Therefore, the revised model determines the maximum time between keying letter i and preparing for letter i+1 and adds it to the total. The $A_0 + I_0$ and $K_{|W|-1}$ terms appear outside the summation, as they apply to the serial boundary cases of the first and last letter, respectively.

3.2.3 Performance Predictions

As an example of applying the revised model, suppose an expert user enters the word "lazy" using the *T9* keyboard. ("lazy" is typically ambiguous and in the phrase, "The

quick brown fox jumps over the lazy dog". Evaluators often use this phrase to evaluate text entry, as it contains every letter in the English alphabet.) Mental preparation (*M*) accounts for 25 ms and initiating the first key press (A_0+I_0) takes 55 ms more. Keying each letter requires more time than initiating the next key press (i.e., $K_i > A_{i+1}+I_{i+1}$ for all *i*). Keying in all four letters takes 1017 ms. Moving the user's gaze to the display costs 70 ms and comparing the displayed word to the intended one (*C*) costs an additional 166 ms. "lazy" requires one press of NEXT ("jazz" appears first), and that takes 319 ms. The user then compares the words again, accepts the displayed word by pressing the SPACE key (269 ms) and compares the displayed and intended words one last time to ensure acceptance of the word. In total, the revised model predicts that entering "lazy" will take 2.25 seconds.

Using software and deriving a word-frequency-keystrokes representation from a corpus can simulate text entry. Table 3-4 through Table 3-6 inclusive represent WPM performance predictions using the BNC1, BNC2, and SMS corpora, respectively. With each corpus, predictions were calculated using the traditional Fitts' performance model (i.e., using "none" of the MHP timings), and this author's revised performance model (i.e., using "expert", "typical", and "novice" MHP timings). The software obtained the average entry time per character (including spaces) by weighting and summing the calculations for all words, and adjusting for word sizes. By multiplying by 5 (the accepted word length for measuring typing speeds) and dividing by 60 (the number of seconds in a minute), the software calculated a WPM performance prediction

(MacKenzie & Soukoreff, 2002b). Cells highlighted in green indicate the highest performing technique in that category. (Highlights also illustrate performance differences hidden by round off.) Between the corpora, the predictions vary little. In addition, the inclusion of MHP timings drastically affects performance. While one-handed *T9* predictions are consistently the lowest, *EQ3* predictions are almost always the fastest. Its QWERTY-styled letter layout distributes keystrokes amongst the keys and minimizes word collisions, thereby reducing the number of presses of NEXT. Furthermore, its small footprint keypad minimizes thumb movement, thereby reducing the required movement time.

Technique	MHP Timings Used			
	None	Expert	Typical	Novice
T9	40.8	31.4	23.7	15.9
T9 (two hands)	62.0	43.9	30.3	18.3
SureType	64.0	44.5	30.6	18.3
EQ3	69.1	46.4	31.5	18.5
EQ6	66.7	45.7	31.2	18.5

Table 3-4: WPM performance predictions using the BNC1 corpus. Green highlights indicate the fastest technique(s).

Technique	MHP Timings Used			
	None	Expert	Typical	Novice
T9	40.9	31.6	24.1	16.2
T9 (two hands)	62.2	44.3	30.8	18.7
SureType	63.8	44.8	31.0	18.7
EQ3	68.7	46.7	32.0	18.9
EQ6	66.6	46.0	31.7	18.9

Table 3-5: WPM performance predictions using the BNC2 corpus. Green highlights indicate the fastest technique(s).

Technique	MHP Timings Used			
	None	Expert	Typical	Novice
T9	40.1	29.6	21.8	14.2
T9 (two hands)	61.4	41.0	27.4	16.1
SureType	62.4	41.5	27.7	16.2
EQ3	68.6	43.6	28.6	16.4
EQ6	66.2	43.2	28.6	16.6

Table 3-6: WPM performance predictions using the SMS corpus. Green highlights indicate the fastest technique(s).

3.2.4 Sensitivity Testing

Sensitivity testing demonstrates how a model's prediction changes with fluctuations in its parameters. A preferred model is one with low sensitivity, where its output changes little with such variations (Card et al., 1980; Card et al., 1983; MacKenzie & Soukoreff, 2002a). Results using the revised model appear in Table 3-7, with each numeric parameter varying independently, and nominal values representing two-handed *T9* entry with the BNC1 corpus and expert MHP timings.

Parameter	Variation					
	50%	80%	90%	110%	120%	150%
Intercepts*	21.40%	7.63%	3.69%	-3.53%	-6.83%	-15.55%
Slopes*	5.52%	2.16%	1.09%	-1.09%	-2.16%	-5.29%
t_{MIN}	6.15%	2.43%	1.21%	-1.28%	-2.56%	-6.25%
t_E	2.35%	0.93%	0.46%	-0.46%	-0.91%	-2.24%
t_P	3.48%	1.37%	0.68%	-0.67%	-1.33%	-3.27%
t_M	3.06%	1.21%	0.60%	-0.59%	-1.18%	-2.92%
t_C	5.20%	2.03%	1.01%	-0.99%	-1.96%	-4.79%
t_S	2.49%	0.98%	0.49%	-0.48%	-0.96%	-2.38%

Table 3-7: Percent change in WPM Prediction when independently varying parameters.
** Left and right parameters varied together.*

While varying the intercepts alters the prediction by up to 21.4%, such a wide variation is unlikely. Variations that are more reasonable reveal a relatively insensitive model. Specifically, the addition of MHP components does not adversely affect sensitivity, as predictions remain within 5.2% of nominal. This maximum deviation occurs when varying t_C, because it is the most contributing MHP component in the revised model. Conversely, the minimum deviation occurs when varying t_E.

3.3 Evaluation Method

This section details the methods employed to evaluate the revised model.

3.3.1 Participants

The Primary Investigator recruited paid participants by posting flyers on the local university campus. Twelve students participated – six males and six females. Ages ranged from 18 to 34 years, with an average age of 24 years. Though all participants chose to use the mouse in their right hand in the right-handed configuration, one participant was actually left-handed. Ten participants used a mouse daily, while two primarily used other input devices. Each participant made an appointment at his or her convenience that lasted approximately one hour and included a questionnaire. The questionnaire gathered demographic information and the participant's self-described experience with text messaging and various input devices.

3.3.2 Apparatus

The workstation for the experiment was a *Pentium 4 530* (3 GHz) system with a 17-inch LCD monitor and a Logitech *Optical Mouse*. To avoid extraneous onscreen stimuli, the Primary Investigator maximized the program window to fill the entire screen and hid the taskbar. The experiment took place in a quiet office environment.

3.3.3 Design

This experiment was a 2 x 5 within-subjects design. The first factor was perceptual and cognitive load with two levels: Present and Absent, detailed subsequently. The conditions were counterbalanced to offset the effects of fatigue and asymmetric transfer of skill (e.g., *Condition A* affecting the result of *Condition B* significantly more than *Condition B* affects the result of *Condition A*).

The second factor was practice, represented by five repetition of flawless input – the test program discarded incorrect entry. The purpose of these repetitions was to familiarize the participant with the layout and to measure his or her learning over time. (The Primary Investigator initially considered ten sessions, but preliminary testing yielded test times in excess of two hours, leaving participants exceptionally fatigued.)

Ten phrases served as input for the text entry tasks and appeared in random order without replacement. The phrases originate from a list compiled by MacKenzie and Soukoreff (2003). The ten phrases possess a cumulative structure that closely matches that of English. The correlation of letter frequency with the corpus is high (98.4%), as is

the correlation of words requiring at least one press of NEXT (89.6%). The following are the ten selected phrases:

- rent is paid at the beginning of the month
- taking the train is usually faster
- what goes up must come down
- the store will close at ten
- have a good weekend
- this is a very good idea
- our fax number has changed
- thank you for your help
- the early bird gets the worm
- the library is closed today

3.3.4 Procedure

This revised model yields a theoretical upper-bound performance prediction. Furthermore, it assumes the user is familiar with the employed letter-key mapping (i.e., zero scan time) and performs flawlessly (i.e., no typographical errors). Consequently, direct comparisons between the predicted and actual performance are impractical using possibly novice participants in a short-term study such as this. Because the introduction of perception and cognition undoubtedly results in a performance decrease in both predicted and actual situations, a feasible alternative was to evaluate the revised model by comparing its predicted relative decrease to the actual relative decrease (both measured in percent). This study simulated a text entry condition with normal perceptual and cognitive load, called the Present Condition, and a condition with minimal load, called

the Absent Condition. Furthermore, the revised model predicted text entry performance under similar conditions. (Timing values of zero simulated performance without perceptual and cognitive load.)

Because of practical difficulties in gathering text entry metrics on actual mobile devices, a workstation, mouse, and onscreen telephone keypad simulated mobile text entry. Figure 3-2 is a screenshot of the test program used. This method reproduces the same perceptual and cognitive components present with mobile device input. Though the involved movements differ, predicting performance for a mobile device would require only the corresponding Fitts' law coefficients for that device.

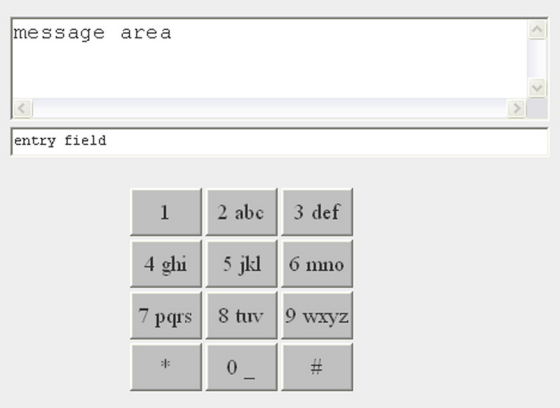

Figure 3-2: A screenshot of the test program used to simulate mobile text entry.

Determining Model Parameters

To ensure accurate model predictions for the experiment, the Primary Investigator measured Fitts' law coefficients for each participant. He used a short pre-test whereby each participant used the mouse to press a single key (e.g., 1) and to alternate between seven key pairs (e.g., 1-2, 1-3, 1-6, etc.) quickly and accurately. As the distance between

key pairs increased, so too did the index of difficulty. For each participant, combining their collected data points with a linear regression yielded corresponding Fitts' law coefficients (MacKenzie, 2003; Silfverberg et al., 2000).

As a heuristic for determining which MHP timings to use, the Primary Investigator categorized each participant based on experience with telephone keypads and text messaging (as gathered from the questionnaire). Table 3-8 lists the assigned values and weights. For example, a participant who used a telephone keypad occasionally (value = 2) and who wrote text messages occasionally (value = 2), would receive a score of 6 (2+2*2) and the classification of "typical".

Experience	**Weight**
Telephone keypad	1
Text messaging	2
Frequency	**Value**
Never	0
Rarely	1
Occasionally	2
Daily	3
Category	**Score**
Novice	[0..5]
Typical	[6,7]
Expert	[8,9]

Table 3-8: Weights, values, and scores used to categorize participants.

Subsequently, the revised model would yield a performance prediction for each participant using his or her Fitts data and the MHP timings associated with his or her classification.

Present Condition

In this condition, a phrase appeared in the message area of the test program. The phrase disappeared once text entry began. This encouraged cognitive processes by simulating the typical entry of a message from memory. Once the participant had memorized the phrase, he or she began by pressing the 0-key (also the SPACE-key) to start the timer and entered the phrase in the text area by using the mouse and onscreen keypad. In reality, entering ambiguous text on a mobile device displays disambiguated words represented by the keystrokes thus far. To avoid such extraneous perceptual stimulation, common practice is to ignore the display until the user reaches the end of the word. The test program enforced this practice by displaying disambiguated words only after the number of keystrokes entered equalled that of the target word.

The Primary Investigator instructed participants to proceed as fast as possible while attempting correct input and paying attention to the correctness of the input. After each word, the participant compared the displayed word to the intended one. At any time, the participant could accept the displayed word by pressing the 0-key, display words represented by the same keystrokes by pressing the *-key, or clear the entry by pressing the #-key. This simulated the actual mental tasks required when using an ambiguous keyboard. If a participant accepted an incorrect word, it disappeared and the participant had to enter the correct word. To encourage participants to attend to the displayed word (instead of relying solely on the keystrokes pressed), the test program randomly (with a

probability of 1/50) inserted a typographical error into the input. The Primary Investigator informed participants of this before starting.

At the end of the phrase, a dialog box appeared encouraging the participant to "take a break". After closing the dialog, the next phrase appeared and input continued as just described. A session ended after the participant entered all ten phrases and the next session began with the same ten phrases in a newly randomized order. The participant's wpm performance represents the best time for each phrase over all sessions.

Absent Condition

In this condition, a single word from an input phrase appeared in the message area for the participant to enter repeatedly five times – once for each session. The immediate repetition aimed to minimize cognitive load. To discourage visual feedback, the word disappeared once text entry began. This was to prevent visual comparison between the entered and intended word. To ensure an equal number of keystrokes in both conditions, ambiguous words appeared with the appropriate number of "*" characters appended. The participant had to enter such characters accurately.

The Primary Investigator instructed participants to proceed as fast as possible while attempting correct input. After memorizing the word, the participant began by pressing the 0-key to start the timer and entered the word in the text area by using the mouse and onscreen keypad. At the end of the word, the participant pressed the 0-key again to end entry for this session and begin entry for the next one. Pressing the 0-key aimed to demarcate the beginning and end of entry, and to simulate the typical preceding

and terminating space character. After five repetitions, a dialog box appeared encouraging the participant to "take a break". After closing the dialog box, the next word in the phrase appeared and input continued. The participant's wpm performance represents the best time for each word over all sessions.

3.4 Results and Discussion

This section presents the results of the evaluation and discusses their implications. Table 3-9 presents the MHP timing category, expertise score, and Fitts data used for each participant.

Participant	Expertise (score)	Intercept (ms)	Slope (ms/bit)
1	Novice (0)	169.00	106.88
2	Novice (2)	122.25	93.00
3	Novice (5)	156.61	103.13
4	Novice (2)	n/a	n/a
5	Typical (6)	162.92	118.98
6	Expert (9)	150.28	110.84
7	Novice (4)	142.27	142.29
8	Novice (3)	121.83	102.59
9	Novice (2)	152.71	73.77
10	Typical (7)	99.82	119.77
11	Typical (7)	107.00	119.32
12	Expert (9)	201.12	95.07

Table 3-9: The model parameters used for each participant.

The Fitts' law data collected from Participant 4 yielded weak linear regression results (R^2 value of 0.08). (Apparently, that participant did not understand the instructions

during the pre-test.) Consequently, the model could not produce reliable results for that participant.

3.4.1 Performance

As expected, perceptual and cognitive load significantly affected performance ($F_{1,10} = 120.58$, $p < .0001$). Participants mean entry rate was 18.0 wpm for the Present condition and 32.0 wpm for the Absent condition. The lower performance results in the Present condition are likely due to the addition of perceptual and cognitive processes to the interaction. For each participant, the Primary Investigator calculated the observed performance and compared this against the revised model's predictions. Impressively, on average, the performance predicted by the revised model (i.e., with perceptual and cognitive loads present) differed from the average observed performance by only 0.5 wpm (3.0%)! Evidently, concerns regarding novice participants and the study's short-term nature were unnecessary. When comparing the decrease in performance corresponding to the addition of perceptual and cognitive loads, as the study intended, the revised model was accurate, on average, within 5%. Detailed results appear in Table 3-10.

Participant	Actual Performance (wpm)			Predicted Performance (wpm)			Difference
	Absent	Present	% Decrease	Absent	Present	% Decrease	(%)
1	30.44	15.18	50.15	33.89	13.89	59.03	8.88
2	33.30	17.91	46.21	42.42	15.09	64.43	18.22
3	32.11	12.27	61.80	35.81	14.20	60.35	-1.44
4*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*
5	33.01	26.81	18.78	32.56	19.91	38.84	20.06
6	34.41	24.08	30.03	35.11	27.28	22.30	-7.73
7	31.63	14.69	53.55	30.97	13.37	56.83	3.28
8	32.74	18.11	44.69	40.15	14.80	63.13	18.45
9	35.77	13.31	62.79	42.72	15.14	64.57	1.77
10	27.54	15.12	45.10	39.23	22.22	43.35	-1.75
11	31.05	19.96	35.73	38.41	21.96	42.83	7.11
12	30.47	20.34	33.23	32.75	25.83	21.11	-12.12
Ave:	32.04	17.98**	43.82	36.73	18.52**	48.80	**4.97****
SD:	2.21	4.54	13.49	4.08	5.11	16.23	**10.76**

Table 3-10: Results of actual and predicted performance.
** Participant 4's Fitts' law data yielded unreliable results.*
*** The intended comparison yields results, on average, within 5%, while direct comparison yields results, on average, within 3%!*

The reality that expert participants did not always yield the best performance illustrates an apparent discrepancy between a participant's self-described experience and his or her actual performance. In one particular instance, the Primary Investigator observed one "expert" spending several seconds searching for the letter "e", the most frequently occurring letter in the English alphabet. Although, in practice, texting vernacular might not employ the letter "e" as often, perhaps a micro-evaluation of a participant's experience prior to the study would yield categorizations that are more accurate.

3.4.2 Learning

Over the five repetitions, participants' performance displayed statistically significant short-term learning effects ($F_{4,40} = 72.26$, $p < .0001$). The effect of condition order proved not statistically significant ($F_{1,10} = 0.003$, ns), verifying effective counterbalancing. Furthermore, there was no asymmetrical transfer of skill between the order of conditions and the conditions themselves ($F_{1,10} = 1.51$, $p > .05$), and none between the order of conditions and the repetitions ($F_{4,40} = 0.60$, ns). On average, participants exhibited an increase in text entry performance of 61.6% and 33.2% and peaked at 28.4 wpm and 15.5 wpm in the Absent Condition and the Present Condition, respectively.

Condition	Repetition 1	Repetition 2	Repetition 3	Repetition 4	Repetition 5
Absent	17.6	25.1	27.3	28.4	27.9
Present	11.7	13.0	14.2	14.6	15.5

Table 3-11: Average WPM performance for each repetition.

In an attempt to forecast the amount of repetition required to achieve the predicted upper bound, Table 3-12 presents the results from employing the power law of learning and the combined model of Isokoski and MacKenzie (2003). However, the large number of trials calculated (some exceeding 1000 trials!) suggests that five sessions is perhaps too few from which to extrapolate long-term performance reliably.

Condition	Power Law	Combined Model
Absent	Approx. 27	Approx. 200
Present (Expert)	> 1000	> 1000

Table 3-12: Repetitions required to achieve the predicted upper bound of performance.

3.5 Summary

By augmenting the traditional text entry performance model with MHP timing values, the revised model more closely reflects actual usage. Because it incorporates perceptual and cognitive loads, the revised model predicts a decrease in expert performance from 41 to 31 wpm for one-handed *T9* input and from 62 to 44 wpm for two-handed *T9* input. It also predicts a decrease from 64 to 45 wpm for *SureType* input, from 69 to 46 wpm for *EQ3* input, and from 67 to 46 wpm for *EQ6* input.

A sensitivity analysis shows the revised model to be reasonably insensitive to variations in its parameters. Altering expert MHP values by up to +/-50% results in prediction fluctuations of 5.2% or less. In addition, the evaluation results show direct performance predictions to be accurate within 3% on average. Participants' performance during five repetitions showed significant learning effects. However, with the use of two separate learning models, five repetitions were not sufficient to forecast the required number of repetitions to reach the predicted upper bound of performance.

With the global proliferation of mobile devices and popularity of text messaging, realistic performance models are important for the evaluation of new device designs and input techniques. This revised model should prove useful when applied to other ambiguous text entry methods where perceptual and cognitive processes combine with movement of the fingers.

Chapter 4
TnToolkit: A Design and Analysis Tool for Ambiguous Keypads

The popular use of mobile phones underscores the significance of mobile text entry. While some phones offer a miniature QWERTY keyboard or speech-to-text input, most provide an ambiguous keypad to allow for discrete input in a small form factor. The prevalence of ambiguous keypads necessitates examination of their performance characteristics. However, existing evaluation tools involve a time-consuming, multi-step process. This chapter presents TnToolkit – this author's self-contained tool to calculate performance measurements for ambiguous keyboards. TnToolkit can calculate a keypad's KSPC, as well as its KCME as outlined in Chapter 2. The toolkit can also calculate a WPM performance prediction based on the traditional Fitts' model (MacKenzie & Soukoreff, 2002a; Silfverberg et al., 2000), or the revised model presented in Chapter 3 of this thesis. TnToolkit calculates aforementioned metrics in a united, rapid, and streamlined manner, while providing additional functionality to the user.

This chapter presents the motivation for and design of TnToolkit. It then describes the features and benefits of the toolkit and details an experiment to evaluate its performance relative to existing programs. This chapter concludes with a presentation of related work by other authors and summarizes TnToolkit's contribution.

4.1 Motivation and Design

TnToolkit is a self-contained, streamlined tool to evaluate ambiguous keypads rapidly. While Tegic Communications' *T9* technology stands for "Text on 9 keys" (T9), this toolkit can evaluate similar techniques that use an arbitrary, *n*, number of keys – hence the name TnToolkit (sometimes abbreviated TnT).

As detailed in Chapter 3, entering text with an ambiguous keypad requires the user to periodically attend to the display and, if necessary, perform actions to disambiguate the intended word. This additional interaction places perceptual and cognitive load on the user for which traditional performance predictions do not account. Previous research by MacKenzie (2002), and MacKenzie and Soukoreff (2002a) involved tools to generate KSPC measurements and WPM performance predictions, respectively. With their permission, this toolkit extends that research.

Using command-line tools to obtain KSPC for a keypad layout is a multi-step process. The basic ingredient is a word-frequency list obtained from a corpus. One constructs the keystrokes to enter each word based on the interaction technique of interest and appends them to the respective entry in the file. Where necessary, the keystrokes include those needed to choose a word in an ambiguous set. The resulting word-frequency-keystrokes file serves as input to an additional utility that calculates KSPC, weighting the keystroke counts for each word by its frequency. To streamline this procedure, this author created `TnKSPC` – a new Java program that encapsulates the process for an arbitrary keypad layout using a *T9*-like disambiguation.

Numerous mobile devices employ ambiguous keypads that encourage two-handed text entry, but the tool provided for predicting WPM performance modeled such input primarily on miniature QWERTY keyboards. This author's novel Java program, TnWPM, models two-thumb text entry on ambiguous keypads, and incorporates Model Human Processor (MHP) (Card et al., 1983) components into the calculation of movement time. By doing so, it simulates the additional perceptual and cognitive load on the user that is involved when using ambiguous keypads and yields a WPM performance prediction that more accurately represents actual usage.

Calculating a WPM prediction using Fitts' law requires the size and location of each key used for text entry. This involves the use of an image of the keypad, from which to create a text-based digitization file. Each line in the file characterizes a key by a space-delimited list of the following values: a unique identifier, an *x*- and *y*-coordinate representing its target point (i.e., where the user must press to activate the key), and its target width (i.e., the maximum radial distance from the target point at which an accurate press can occur) (MacKenzie & Soukoreff, 2002a). This typically necessitates the use of a graphics application that displays the coordinate location of the mouse pointer. The user then opens the image within the application and uses the mouse pointer as a probe to determine the coordinates of a key. However, such applications are difficult to obtain, time-consuming to install, or poorly suited for the task. To combat this, this author wrote additional Java classes that provide an innovative graphical user interface (GUI) to facilitate digitization tasks. Combined with TnKSPC and TnWPM, they form TnToolkit. (For

further details regarding the Java classes comprising the TnT package, please refer to Appendix B.)

4.2 Features and Benefits

Those wishing to use TnToolkit can download its distributable package from www.cse.yorku.ca/~stevenc/TnToolkit/. In addition, Appendix A provides details on running TnToolkit and on its file structure.

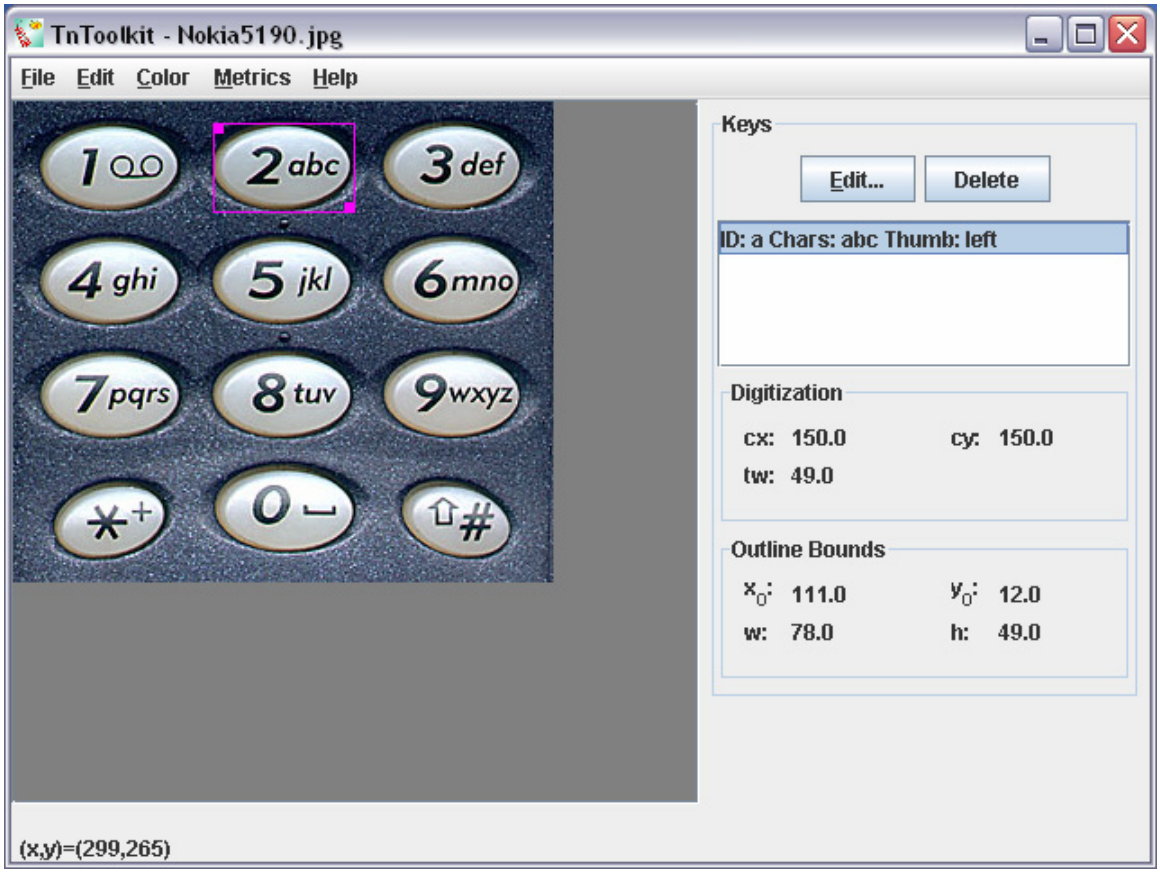

Figure 4-1: TnToolkit's main screen. The user has already digitized and selected a key.

Upon starting the toolkit's GUI interface (Figure 4-1), the user can select `Open` from the `File` menu to load an image file (with extension `gif`, `jpeg`, `jpg`, or `png`) representing the keypad to be analyzed. By selecting "TnT Workspace file" as the file type, the user can retrieve previously saved work. The user can save work in progress at any time by selecting `Save` or `Save As` from the `File` menu.

4.2.1 Keypad Digitization

Once a keypad's image is loaded, the user performs the digitization of a key by simply dragging across regions within the image. Simultaneously, the status bar at the bottom of the window displays the mouse's current location, as well as the dimensions of the outline. The Key Definition dialog (Figure 4-2) then appears to facilitate mapping. The user indicates whether the key represents letters, or the NEXT or SPACE functionalities. (Note that some mobile phones label the NEXT- and SPACE-keys differently.) For letter keys, the user selects the checkboxes that correspond to the letters associated with that key. Once the user confirms the key's attributes (by pressing `OK`), the checkboxes for the mapped letters are disabled. This prevents invalid mappings by restricting the user to make valid selections only. The dialog displays a default identifier for that key, but the user can replace it with a different, but unique, character. Finally, the user stipulates that one typically presses that key with the left thumb, right thumb, or that it is equally accessible to both thumbs. Though this thesis uses the term "thumb", such input also includes other two-handed methods such as using a finger on each hand, or using two

styli simultaneously. Additionally, assigning all keys to a single thumb can simulate input on traditional, one-handed devices.

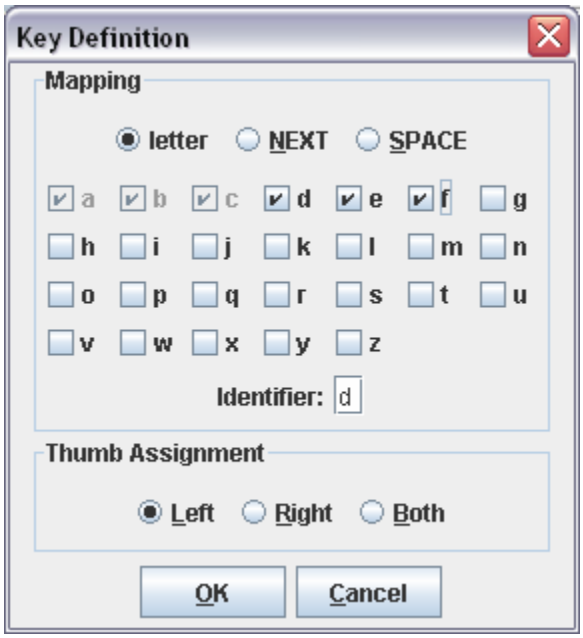

Figure 4-2: The Key Definition dialog to facilitate key-letter mapping. It disables previously mapped letters to prevent invalid mappings.

Digitized keys appear as entries in the list on the right of the window. The user can select a key by clicking its entry in the list or by clicking within its outline. Once selected, the user can delete the key or edit its mappings by pressing the `Delete` or `Edit` button, respectively. In addition, the attributes of the selected key appear along the right of the window. The values x_0 and y_0 represent the top left point of the key's outline, and the values `w` and `h` represent its width and height, respectively. The key's target width (`tw`) is the smaller of its width and height, and the key's target point is its center point, represented by `cx` and `cy`. This form of digitization is consistent with previous tools, but

it can inaccurately represent an irregularly shaped key whose target point is not at its center. This heuristic also poorly represents keys that are especially long. For example, a long but narrow SPACE-key located in the middle of the keypad might benefit from the definition of two target points, each typifying activation by a separate thumb. The ability to define custom or multiple target points would result in increased metric accuracy for keypads with unusually shaped keys. While no tool currently implements this functionality, TnToolkit's GUI would facilitate and utilize it more easily than the previous command-line programs.

To ensure accurate WPM performance predictions, a key's outline should represent a bounding rectangle – the smallest possible rectangle that encompasses the entire key. By first selecting a key's outline, the user can the drag the top-left handle (i.e., a small square attached to the outline) to move it, or drag the bottom-right handle to resize it. To make the outlines clearly visible, the user can change their colour via the `Color` menu; the user can select a preset colour, or choose a custom one. In addition, the user can undo or redo changes to the workspace by selecting `Undo` or `Redo`, respectively, from the `Edit` menu.

4.2.2 Setting Parameters

By selecting `Parameters` from the `Metrics` menu, the user can modify model parameters, such as the word-frequency file, Fitts' law coefficients, and MHP timings as described in Chapter 3.

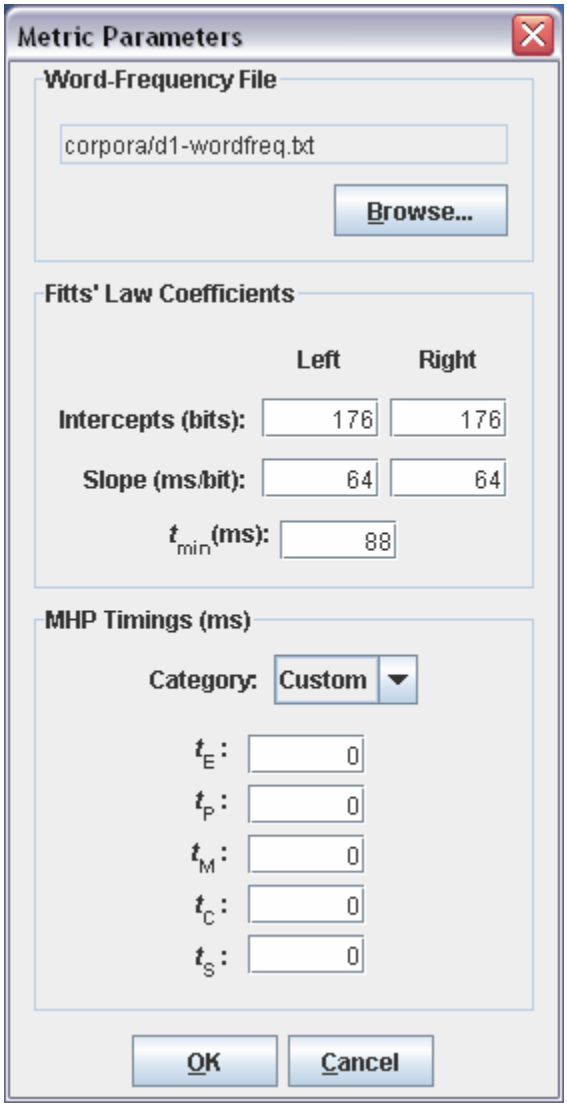

Figure 4-3: The Metric Parameters dialog.

The user can select preset values for an expert, typical, or novice user, or set custom values. To discount the effect of perceptual and cognitive effort (i.e., to generate a traditional upper-bound expert prediction) the user can set custom values of zero for all timings.

4.2.3 Exporting Data

TnToolkit still contains command-line programs to calculate performance metrics (i.e., `TnKSPC` and `TnWPM`), but these tools can be used in conjunction with the GUI interface. By selecting `Export` from the `File` menu, the user can export the digitization, mapping, and parameter data defined within the GUI. The resulting text files can then serve as input for TnToolkit's command-line interface, or the previous command-line tools. This flexibility allows for scripting, execution via a telnet terminal, and compatibility with input files created for the previous command-line tools.

4.2.4 Calculating Metrics

When the user has mapped all letters and functions to keys, the user can calculate performance metrics by selecting `Calculate` from the `Metrics` menu. If the user forgot to perform all the required mappings, a dialog indicates which letters or functions remain. Otherwise, a dialog presents the progress of the calculations and allows the user to halt the process by pressing `Cancel` (Figure 4-4).

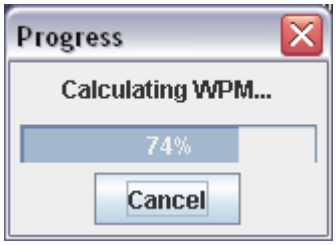

Figure 4-4: The Progress dialog shows calculation progress. It also allows the user to halt the process.

Typically within several seconds, the results dialog appears. The `KSPC Data` panel (Figure 4-5) displays values for KSPC26, which represents usage of only the 26 letters of the English alphabet, the more practical KSPC27, which includes the space character, and ambiguous word statistics. It also displays a result for the KCME metric detailed in Chapter 2.

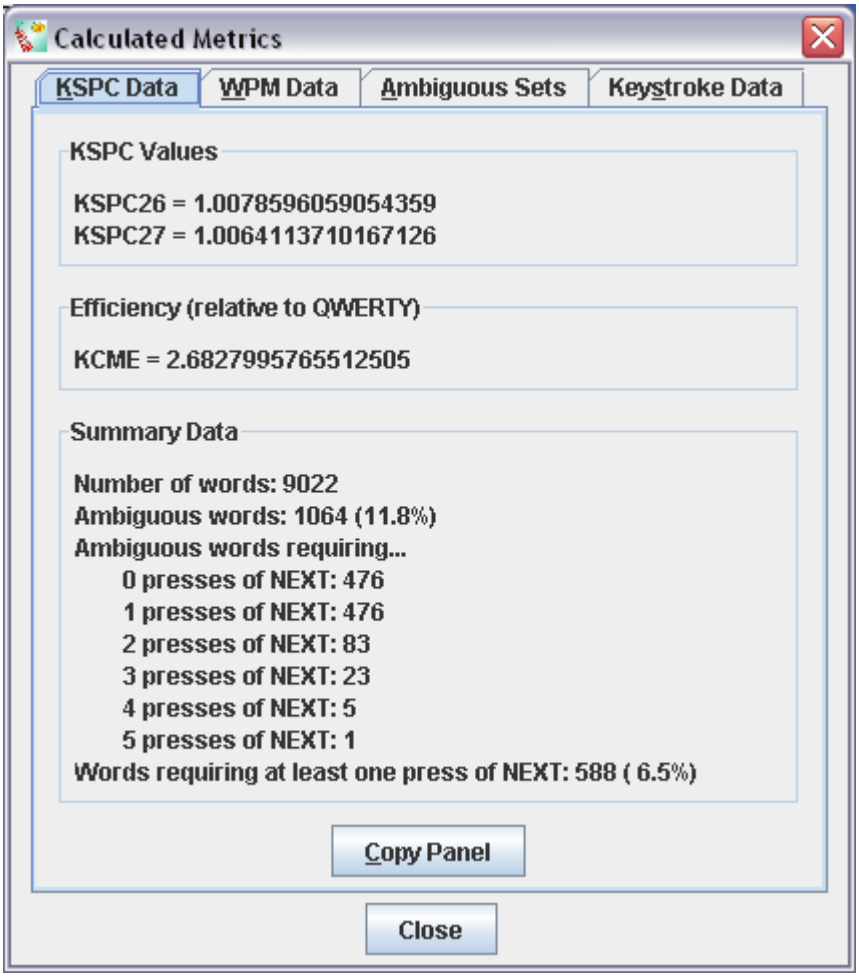

Figure 4-5: The KSPC data calculated by TnToolkit.

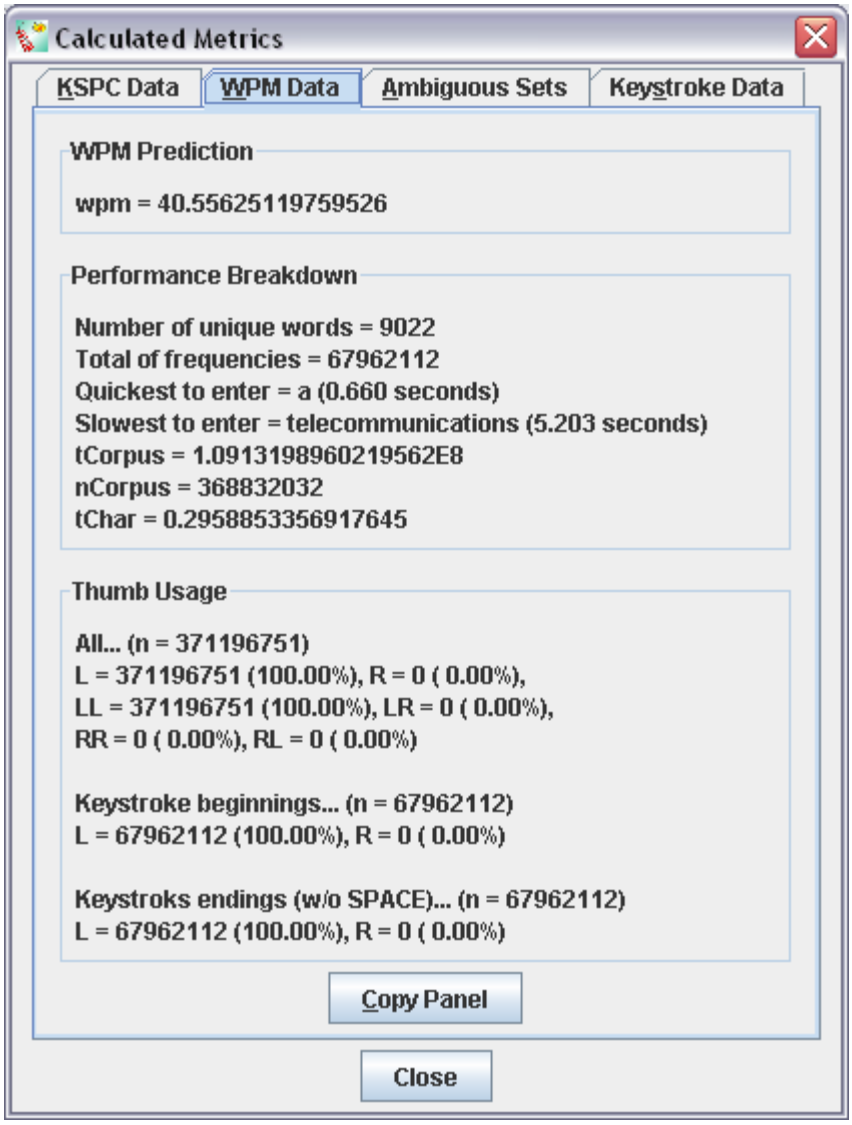

Figure 4-6: The WPM data calculated by TnToolkit.

In addition to a WPM performance prediction, the `WPM Data` panel (Figure 4-6) presents thumb usage statistics, and itemizes characteristics, such as the estimated time to enter all the words in the word-frequency file (`tCorpus`), the number of characters necessary for such a task (`nCorpus`), and the average time to enter each character (`tChar`). The results dialog also contains panels that list ambiguous word sets (Figure

4-7) and keystroke data (Figure 4-8) applicable to the current keypad layout. To facilitate further analysis or alternate presentation of this data, the user can copy the contents of each tabbed panel to the clipboard and paste it into other applications.

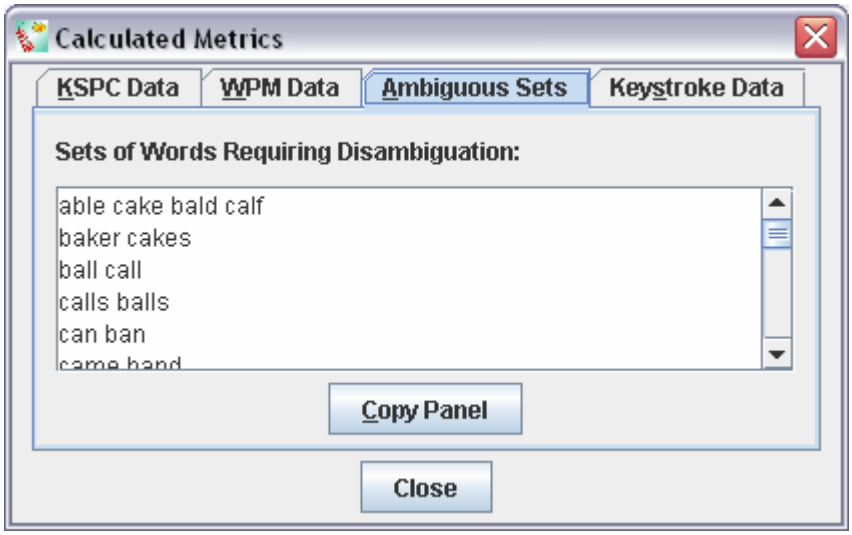

Figure 4-7: Associated ambiguous word sets.

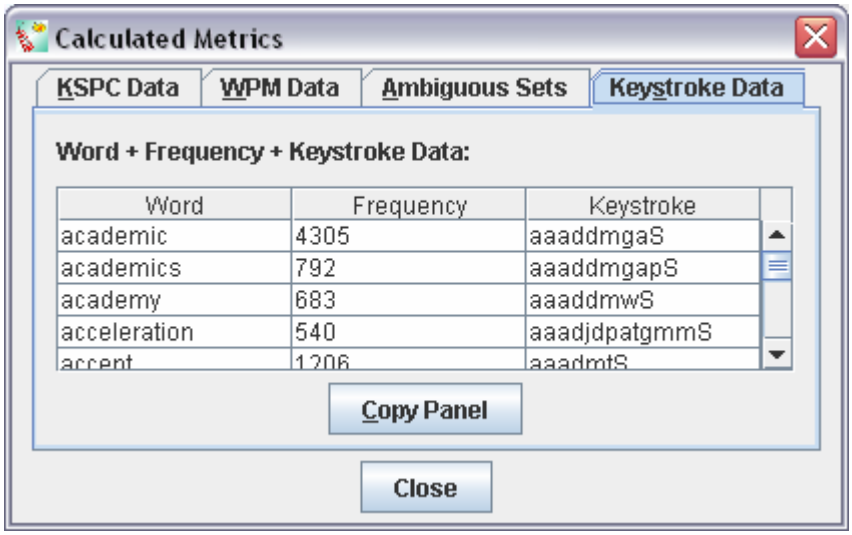

Figure 4-8: Associated keystroke data.

4.2.5 HTML-Based Help Files

To assist users, the TnT distribution includes HTML-based help files, which users can view in a mini browser from within TnToolkit (Figure 4-9). Accessed from the `Help` menu or by pressing `F1`, the help files guide detail how to manage files, digitize keypads, and calculate metrics. It also assists users by describing the metrics used and how to interpret their results.

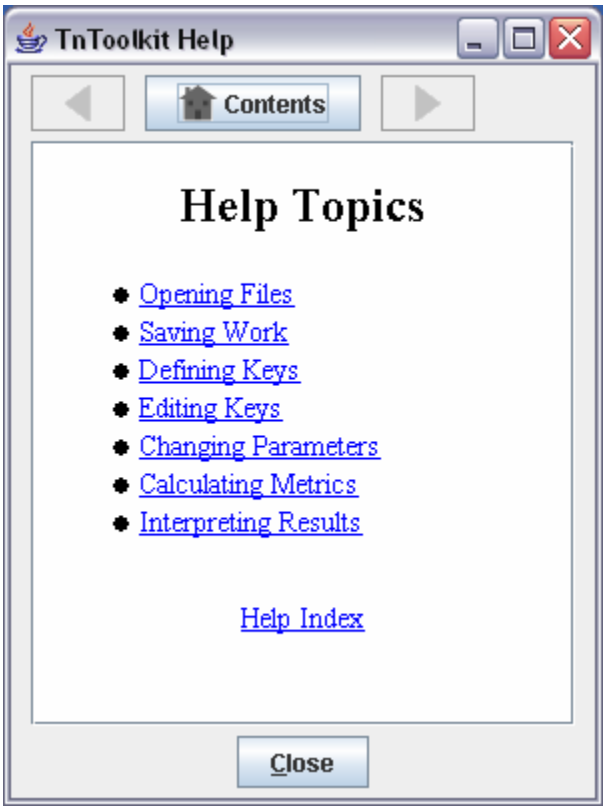

Figure 4-9: Packaged HTML-based help files, viewable from the `Help` menu.

4.3 Evaluation Method

Since the expeditious modeling of existing or hypothetical designs is a central motivation, an empirical evaluation of TnToolkit compares its performance to existing tools. The methodology and results of that evaluation is the topic of this section.

4.3.1 Participants

Primary Investigator recruited volunteers from the department of Computer Science and Engineering. Twelve students participated – ten males and two females. Ages ranged from 22 to 38 years, with an average age of 27 years. Though all participants opted to use the mouse in their right hand in the right-handed configuration, two participants were actually left-handed. Each participant made an appointment at his or her convenience that lasted approximately thirty minutes. None had any previous experience with TnToolkit.

4.3.2 Apparatus

The workstation for the experiment was a *Pentium 4 530* (3 GHz) system running *Windows XP*. It used a 17-inch LCD monitor and a Logitech *Internet Keyboard* and *Optical Mouse*. The experiment took place in a quiet office environment.

4.3.3 Design

This experiment was a single factor design. The within-subject factor is Interface Type with two levels: Command-Line versus TnToolkit (GUI). The Primary Investigator

counterbalanced their order to offset learning interference of one condition on the other. He measured two dependent variables: Task Completion Time and Result Accuracy.

4.3.4 Procedure

Participants were given the task of calculating KSPC measurements and a WPM performance prediction for a given keypad image. To allow for comparisons with established results, this study employed the same keypad, word-frequency file, and Fitts' law coefficients used by Silfverberg, MacKenzie and Korhonen (2000). Each participant performed the task once in each condition, described subsequently. In each case, the Primary Investigator presented participants with a written script to guide them through each condition. He explained the task and answered questions. The primary investigator started timing when the participant indicated readiness and stopped timing as soon as the participant finished writing down the aforementioned measurements.

Command-Line Condition

In this condition, participants used the command-line programs by MacKenzie (2002) and MacKenzie and Soukoreff (2002a) to calculate KSPC measurements and a WPM performance prediction, respectively.

The first step was to create a keypad digitization file. For this study, keys 2-9 represented their corresponding letters, and "N" and "_" represented the NEXT (*-key) and SPACE (0-key) keys, respectively, as required by the program. In addition, a key's target point was its center point, and its target width was the minimum of its width and

height. Participants used *Microsoft Paint* (and *Calculator* if necessary) to determine coordinate values, and *Notepad* to create the digitization file. Instead of approximating the center point of a key, participants determined the edge coordinates of each text entry key. They then calculated the center x-coordinate to be average between the left- and right-edge coordinates, and the center y-coordinate to be the average between the top- and bottom-edge coordinates.

Participants then executed three programs at the command-line. The first program read the word-frequency file and produced a word-frequency-keystrokes file by replacing each letter in a word by the corresponding number on a telephone keypad. The second program appended disambiguating keystrokes in accordance with *T9*-like input. The third program yielded values for KSPC26 and KSPC27.

Calculating a WPM performance prediction required the participants to enter the keypad digitization filename, the word-frequency-keystrokes filename, and the text entry keys' identifiers into a model definition file. Upon saving changes to the text file, participants then used it as input to a command-line program that yielded a WPM performance prediction.

TnToolkit Condition

In this condition, participants launched TnToolkit and opened the image of the keypad. The script instructed them to define the letter keys (keys 2-9), the NEXT-key (*-key), and the SPACE-key (0-key), assigning all of them to the same thumb. It also informed participants of the ability to edit a key. After defining the required keys, the participants

selected `Calculate` from the `Metrics` menu to calculate and display the KSPC measurements and WPM performance prediction.

4.4 Results and Discussion

All participants appreciated the convenience afforded by TnToolkit. Some believed its use would reduce errors and make any errors that did occur easier to identify and correct. However, one participant preferred the control permitted by the command-line programs, and suggested the use of more handles in the toolkit with which to move and resize a key's outline.

4.4.1 Task Completion Time

As shown in Table 4-1, Interface Type had a significant effect on Task Completion Time ($F_{1,10} = 25.88$, $p < .0005$). On average, Task Completion Time was 69% quicker using TnToolkit than using the command-line programs!

Participant	Task Completion Time (*mm:ss*)		
	Command-Line	TnToolkit	% Decrease
1	24:56	06:11	75.20
2	12:10	03:23	72.19
3	12:50	04:38	63.90
4	43:29	03:40	91.57
5	17:39	05:25	69.31
6	09:37	05:56	38.30
7	11:57	04:25	63.04
8	20:09	05:12	74.19
9	13:25	04:18	67.95
10	26:50	05:08	80.87
11	18:09	05:25	70.16
12	32:16	10:49	66.48
Ave:	20:17	05:23	69.43
SD:	09:37	01:50	12.03

Table 4-1: The results for Task Completion Time.
Green highlights indicate the faster condition.

In addition, statistical analysis confirms effective counterbalancing ($F_{1,10} = 0.044$, ns) and that no asymmetric transfer of skill occurred between the two conditions ($F_{1,10} = 0.013$, ns).

4.4.2 Result Accuracy

Because both the key-letter mapping and the word-frequency file remained the same throughout the experiment, all participants obtained the same KSPC measurements in both conditions. Unpublished statistics by MacKenzie verify the KSPC26 value of 1.0079 and the KSPC27 value of 1.0064. (MacKenzie's published results (2002) differ by less than 0.08%, but were calculated using a different word-frequency file as input.)

The Fitts' law model underlying the WPM prediction requires the user to provide the bounds (i.e., size and location) of each key used for text entry. Shadows in the keypad's image can make the determination of a key's bounds subjective, resulting in slight variations in predictions. Table 4-2 presents the WPM predictions calculated by each participant in each condition. It also presents the percent difference from the accepted value of 40.6 wpm for this keypad calculated by Silfverberg, MacKenzie and Korhonen (2000). Again, statistical analysis confirms effective counterbalancing ($F_{1,10} = 1.32$, $p > .05$) and that no asymmetric transfer of skill occurred between the two conditions ($F_{1,10} = 3.19$, $p > .05$).

Participant	Command-Line		TnToolkit	
	Prediction	% Difference	Prediction	% Difference
1	45.5	12.07	39.9	1.72
2	41.2	1.48	40.3	0.74
3	40.5	0.25	41.2	1.48
4	38.6	4.93	41.6	2.46
5	38.5	5.17	41.1	1.23
6	41.1	1.23	40.1	1.23
7	41.9	3.20	41.6	2.46
8	41.2	1.48	41.6	2.46
9	41.3	1.72	39.5	2.71
10	40.9	0.74	39.7	2.22
11	41.1	1.23	40.5	0.25
12	40.8	0.49	41.1	1.23
Ave:	41.1	2.83	40.7	1.68
SD:	1.7	3.33	0.8	0.78

Table 4-2: The WPM predictions and their accuracy. The established prediction is 40.6 wpm (Silfverberg et al., 2000). Green highlights indicate the more accurate condition, while yellow highlights indicate both conditions are equally accurate.

Seven of twelve participants achieved the same or better accuracy with TnToolkit than with the command-line programs. Furthermore, all participants attained a prediction within 3% of the accepted value by using the toolkit. However, such differences were not statistically significant ($F_{1,10} = 1.73$, $p > 0.05$). While this indicates that using the toolkit does not necessarily reveal more accurate results than the command-line programs, it also suggests that using TnToolkit is just as accurate as the much slower alternative.

4.5 Related Work

Composed of goals, operators, methods, and selection rules, the GOMS model proposed by Card, Moran, and Newell is the standard for predicting performance and analyzing interaction between humans and computer systems (Tollinger et al., 2005). However, in practice, designers rarely use GOMS modeling, as the overhead required to produce such models often eclipses their benefit (Tollinger et al., 2005).

Tollinger et al. combined GOMS with a model of human cognitive architecture to produce X-PRT, an environment to support interface design and performance evaluation (Tollinger et al., 2005). In general, X-PRT simplifies the process by allowing users to create large systems by combining smaller components that are predefined, user-defined, or imported. For example, they define interface screens by using drawing tools, or by importing an existing image of the screen and outlining the interactive widgets. Users employ primitive operations to define tasks, which they can then combine and structure hierarchically. And with the use of a slider, they can define the simulated user's skill.

This in turn selects the corresponding built-in cognitive model. Other cognitive architecture parameters can also be imported.

Like X-PRT, TnToolkit is the amalgamation of multiple modeling and evaluation tools, it allows users to import existing images to aid interface definition, and it can simulate users of various skill levels. However, while X-PRT is applicable to a broad spectrum of interface types, TnToolkit specializes on interfaces for mobile text entry. Consequently, it is more straightforward and suitable for that task.

An evaluation tool by Sandnes is also specific to text entry. However, unlike Tollinger et al., Sandnes employs a methodology other than GOMS and models text entry techniques using finite state automata (Sandnes, 2005). Traversal algorithms can then evaluate the resulting directed graphs. Specifically, they can calculate values for KSPC and Sandnes' own mean error recovery distance (MERD) metric. While such values can be determined early in a technique's design, the lack of spatial details precludes prediction of WPM performance. In contrast, TnToolkit calculates both WPM and KSPC metrics.

4.6 Summary

With the global proliferation of mobile devices and popularity of text messaging, readily obtainable performance measurements are important for the evaluation of new device designs and input techniques. TnToolkit rapidly and accurately analyzes the performance characteristics of ambiguous keypad layouts. It streamlines performing key-letter assignments and simplifies digitizing ambiguous keypads. Furthermore, it allows users to

visualize and easily edit the keypad digitization, save work in progress, and share data with other applications.

The conducted experiment revealed that use of TnToolkit resulted in a 69% decrease in task completion time. While a reduction in task completion time is a common result of adding a GUI, such an immense improvement *without compromised accuracy* is very much a benefit.

By conveniently facilitating the performance evaluation of ambiguous keypad designs, TnToolkit facilitates analyses of existing devices as well as new prototypes.

Chapter 5
Conclusion

With a multitude of functionality, mobile phones are both ubiquitous communication tools and versatile entertainment units. To take full advantage of their potential, text entry on such devices is paramount. Furthermore, input techniques employing ambiguous keypads seem the methods of choice, as they provide the needed functionality without sacrificing mobility.

Presented in Chapter 4 of this thesis, TnToolkit facilitates the evaluation of existing techniques and the design of new ones. In addition to allowing users to save, export, and share keypad data and metric results, TnToolkit simplifies keypad digitization and includes numerous performance and efficiency metrics. In addition to traditional metrics, it also incorporates the research presented in Chapter 2 and Chapter 3 of this thesis. The new KCME efficiency metric in Chapter 2 combines a technique's KSPC value with the number of keys it employs for text entry. By doing so, it yields a value that represents the technique's balance between QWERTY practicality and device portability. To evaluate a technique's predicted performance, the model in Chapter 3 uses MHP timing values to account for time spent on perceptual and cognitive tasks. Ambiguous text entry requires the user to expend perceptual and cognitive effort to disambiguate entry and ensure that the inputted word is the desired one. By incorporating additional

operators, the author's revised model accounts for this and more closely resembles actual use.

5.1 Future Work

While the author of this thesis has brought to fruition the concepts presented herein, development is a continuous process. The evaluations detailed in Chapter 3 and Chapter 4 helped to assess the model and tool presented in their respective chapters. They also provided the forum for improvements to those endeavours. The subsequent subsections describe such enhancements.

5.1.1 Efficiency Metric

To avoid a bias towards an input technique's KSPC27 value or the number of text entry keys it requires (i.e., *numKeys*), the KCME metric gives equal weight to each measure. However, Table 2-1 indicates that *numKeys* values occupy a much larger range than KSPC27 values. Consequently, *numKeys* unintentionally dominates the metric's calculation. Future development of this metric could investigate including coefficients to attenuate the contribution of *numKeys*, intensify the effect of KSPC27, or both.

5.1.2 Performance Model

While the pre-study questionnaire gathered a participant's self-described telephone keypad and text messaging experience, this data was naturally subjective. Its aim was to categorize participants based on their familiarity with the letter-key mapping employed in

the study. To that end, future evaluations of a similar nature could administer a brief and simple pre-test: the Primary Investigator would present each participant with an illustration of the keypad used (void of markings), and instruct him or her to label the keys appropriately. Thus, the Primary Investigator could categorize participants based on a quantitative analysis of their familiarity with the technique's layout.

Instead of simulating the mobile phone interface on a workstation, the Primary Investigator could conduct the evaluation on an actual mobile phone. The Java 2 Mobile Edition (J2ME) platform enables a subset of desktop Java (J2SE) functionality on devices with limited computational resources (i.e., mobile devices). While many contemporary mobile phones implement a J2ME runtime environment, most have limited support. For example, some cannot support `float` or `double` data types, precluding calculation of Fitts' law coefficients; and the majority cannot write to persistent storage, ruling out logging of performance details. However, mid- to high-end phones now implement floating-point calculations and allow J2ME applications to read from and write to internal flash memory, thus alleviating the aforementioned issues.

Though hardware now makes running a model evaluation feasible, porting the existing program to the J2ME platform would not be easy. J2ME requires different development kits and tools (e.g., J2ME emulator for the desktop environment) than J2SE. It also provides different packages and classes. Some required classes are missing (e.g., `StringTokenizer`, used for parsing input files), so new classes must implement the missing functionality. However, the most significant alteration would be to the GUI.

Because mobile phones do not provide the same input methods as desktop computers (e.g., no mouse), J2ME provides classes to implement GUI components that are different than, and completely incompatible with, J2SE. Program start-up, event handling, and program termination are also very dissimilar, and would require particular management.

5.1.3 TnToolkit

While TnToolkit allows users to digitize a keypad simply by using an image of it, the appearance of the image can affect the accuracy and experience of digitization. Especially small images can obscure a key's edge, and particularly large images might require a great deal of scrolling. To combat this, future versions of TnToolkit could provide zooming functionality. In addition to scaling the image, determination of a key's size and location must also compensate for the image's new size. Failure to do so would detrimentally misrepresent the relative size and distance measure required by the Fitts-based performance model, and yield in drastically accurate predictions. By multiplying coordinates by the scaling factor, digitization results would consistently reflect the original dimensions of the keypad's image. This would also allow the user to make magnification changes at various times during digitization without corrupting the results.

As suggested by a study participant, the addition of more move and resize handles would allow greater control of digitization. However, currently, move and resize handles appear as filled squares. Consequently, they sometimes mask the corners and edges of a key, making accurate digitization difficult; increasing their number would further obscure the user's view of the digitized key. Implementing an increased number of *outlined*

handles would address both issues. By drawing handles that appear as outlined squares (i.e., with no fill colour), additional move and resize handles would provide greater digitization control without hindering the user's view of the keypad.

Sometimes, keypads employ especially large or irregularly shaped keys. When digitizing a key, TnToolkit assigns it a single target point at the center of the key. While this practice is sufficient for traditionally uniform keys, it does not accurately reflect peculiar ones. Future versions of TnToolkit could remedy this by assigning a single target point at the center of a key by default, and allowing users to edit its location and add supplementary ones. This would reveal more accurate performance predictions for keypads with large or irregularly shaped keys.

Appendix A
Running TnToolkit

TnToolkit is available from www.cse.yorku.ca/~stevenc/TnToolkit/. It requires an unzip utility (e.g., 7-zip, WinZip, etc.) to extract it, and the Java Runtime Environment (version 1.5.0 or later) to run it.

A.1 File Structure

Extracting TnToolkit yields the following files and directories:

corpora: Word-frequency representations of corpora appear in this directory.

doc: Javadoc files associated with TnToolkit are in this directory.

help: Users can access TnToolkit's HTML-based help files in this directory.

keyboards: This directory contains default images of mobile phone keypads.

License.txt: Before using TnToolkit, users should read this User License.

TnT.cmd: This start-up script is for the Windows NT operating system or later.

TnT.ico: In Windows, this icon file can be associated with a shortcut to the TnToolkit start-up script.

tnt.jar: This archive contains the class files and resources required by TnToolkit.

TnT.sh: This start-up script is for Linux/Unix operating systems.

A.2 Running the GUI

To launch TnToolkit's GUI, simply run the file `TnT.cmd` (for Windows) or `TnT.sh` (for Linux/Unix). Alternatively, one can enter the following at a command prompt:

```
PROMPT>java –jar TnT.jar
```

A.3 Using the Command-Line Tools

Though users can realize the most benefit from TnToolkit by using its GUI, its command-line tools allow it to function in command-line environments. The subsequent subsections detail how to use these tools.

A.3.1 TnKSPC

This tool takes a letter-key mapping and a word-frequency corpus representation and calculates KSPC-related metrics. Invoking it with insufficient parameters displays the following usage message:

```
PROMPT>java TnKSPC mapping wordfreq [-e] [-s] [-a] [-k]

   where:
      mapping  = file containing letter-key mapping
      wordfreq = file containing word and frequency values
      -e = outputs KCME value
      -s = outputs summary data
      -a = outputs ambiguous word sets
      -k = outputs word-freq-keystroke data

Default output is KSPC26 and KSPC27 values only.

See JavaDoc for more information.
```

As indicated, this tool takes two file paths as input. The first file represents the letter-key mapping of the keypad under analysis. This file must contain two lines. The first lines must contain all the letters of the alphabet pertaining to the target language. If applicable, the letters must be in lower case. The second line should contain each key's identifying character. These characters must be ordered (and repeated if necessary) such that each identifying character appears in the same column as the letter(s) mapped to it. (The user can also export this data from the TnToolkit GUI.) The following example represents a standard telephone keypad:

```
abcdefghijklmnopqrstuvwxyz
22233344455566677778889999
```

The second file provides a word-frequency representation of a corpus. (For tools to assist in the creation of such a file, refer to work by MacKenzie and Soukoreff (2003).) Each line must be a white space delimited list of a word in lower case and its frequency. For example:

```
...
able    26890
bald    569
cake    2256
calf    561
...
```

When a user specifies the `-k` option, this tool outputs the contents of the word-frequency file with each word's required keystrokes appended to its entry. By redirecting this output to a text file, a user can create a word-frequency-keystrokes file, which the `TnWPM` tool requires.

A.3.2 TnWPM

This tool takes a model definition and yields a WPM performance prediction based on Fitts' law. Invoking it with insufficient parameters displays the following usage message:

```
PROMPT>java TnWPM model.txt [-b] [-d] [-m] [-t]

   where:
      model.txt = a model definition file
      -b = breakdown of prediction
      -d = debug information
      -m = model components and parameters
      -t = thumb usage statistics

Default output is WPM prediction only.

See JavaDoc for more information.
```

To minimize the number of parameters entered on the command-line, this tool reads all model parameters from a separate model definition. This text file contains thumb assignment, Fitts' law coefficients, and MHP timing values. The following is a sample of such a file (comment lines begin with '#'):

```
# word-frequency-keystrokes file
d2-ST_ksfreq.txt
# keyboard definition file
7100t_digitization_v2.txt
# left thumb letter assignments (leave blank if none)
Q12A45Z78N
# right thumb letter assignments (leave blank if none)
23P56L89N
# prefer left thumb (when both are equally applicable)?
false
# Fitts' law coefficients...
# left thumb intercept
176
# left thumb slope
64.0
```

```
# right thumb intercept
176
# right thumb slope
64.0
# tMIN (minimum inter-key stroke time using opposite thumbs)
0.088
# Space key policy...
#    'Alternate' = alternate thumb for space at end of word
#    'Left' = always left thumb
#    'Right' = always right thumb
Alternate
# Values for perceptual, cognitive and motor processes...
# Eye movement time (t sub E) in seconds
0.070
# Perceptual processor cycle time (t sub P) in seconds
0.050
# Motor processor cycle time (t sub M) in seconds
0.030
# Cognitive processor cycle time (t sub C) in seconds
0.025
# Time in seconds to determine if two words are the same
0.036
# *** end ***
```

However, the two most complex parameters in the definition are the paths to the word-frequency-keystrokes file and the keypad digitization file. Each line in the word-frequency-keystrokes file lists a word in the corpus, its frequency within the corpus, and the keystrokes required to enter it. Though a user can manually create such a file, the `TnKSPC` tool can also generate it. The following is an excerpt from such a file:

```
...
able     26890    2253S
cake     2256     2253NS
bald     569      2253NNS
calf     561      2253NNNS
...
```

Each line of a keypad digitization (a.k.a. definition) file defines the characteristics of particular key used for text entry. The token on a line is a unique character identifying the key. Key identifiers must correspond to those used to encode the keystrokes in the word-frequency-keystroke file. The next token is a string composed of the letters mapped to that key. Though not strictly required to calculate a performance prediction, this string provides means for error checking within the TnToolkit GUI and is required for compatibility reasons. It also allows the digitization file to be easily comprehendible by humans. The next three tokens represent the key's x-coordinate, y-coordinate, and target width, respectively. Because this tool determines the distance between two keys using the Pythagorean Theorem, neither the location of the coordinate grid's origin nor the unit of measurement is significant. Typically, "S" and "N" identify the SPACE- and NEXT-keys, respectively and other key identifiers are all lowercase. (The user can also export a keypad's digitization data from the TnToolkit GUI.) The following represents a sample keypad:

```
S   S     13.25   27   6
2   abc   13.25   0    6
3   def   26.5    0    6
4   ghi   0       9    6
5   jkl   13.25   9    6
6   mno   26.5    9    6
7   pqrs  0       18   6
8   tuv   13.25   18   6
9   wxyz  26.5    18   6
N   N     25.5    -9   6
```

Appendix B
Primary TnToolkit Classes

This appendix details the primary classes that comprise TnToolkit and their interaction with other classes in the TnT package. (For simplicity, it omits some subordinate classes.) To access individual classes in the TnT package, simply add its path to the system's `CLASSPATH` variable. For further information on a particular class, including inherited features, method details, and version information, please refer to the packaged Javadoc documentation. To give an overview of TnToolkit's design, Figure B-5-1 depicts its UML class diagram.

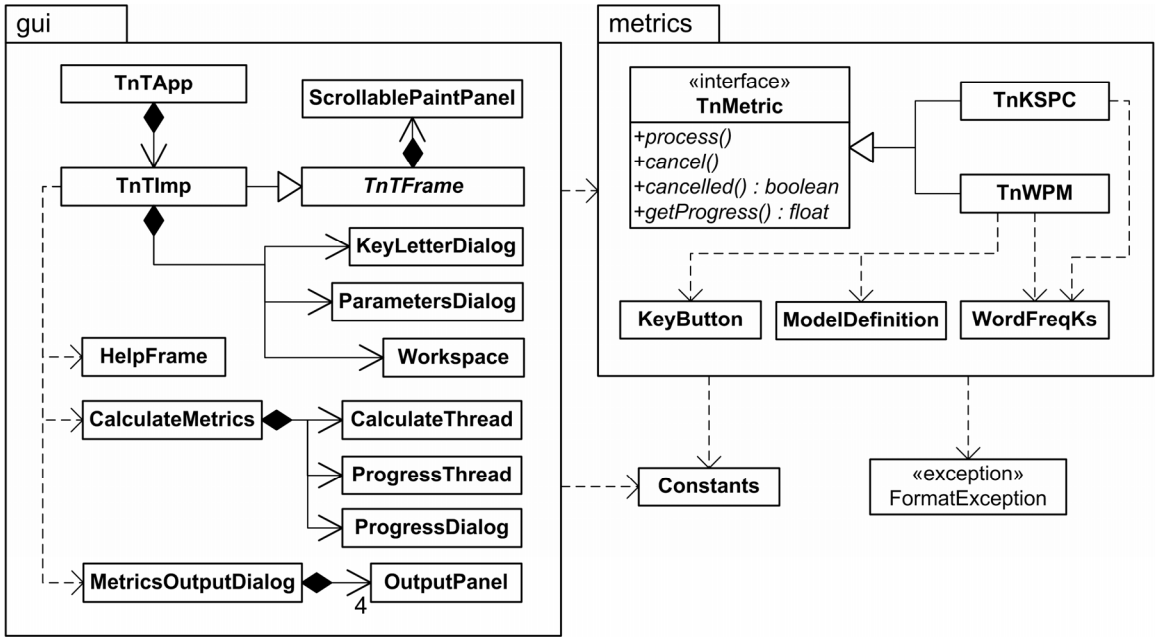

Figure B-5-1: A UML class diagram illustrating the design of TnToolkit. For simplicity, it omits some subordinate classes.

B.1 `tnt.*`

Classes within all subpackages of the TnToolkit project typically use the upper-level classes in this section.

B.1.1 `Constants`

This class defines constant values for the entire project. Such values include defaults and version information

B.1.2 `FormatException`

When reading formatted text files (e.g., word-frequency files, saved workspace files, etc.), methods that encounter discrepancies between the actual and required format should throw an instance of this class. Furthermore, the exception's message should describe the specific cause or location of the format error.

B.2 `tnt.metric.*`

This package provides classes for the calculation of performance metrics, such as KSPC and WPM. It is also the source for TnToolkit's command-line tools: `tnt.metric.TnKSPC` and `tnt.metric.TnWPM`.

B.2.1 `KeyButton`

Instances of this class represent the digitized keys/buttons on the ambiguous keypad. At least five arguments define each key: a character used to identify this key when listing

keystrokes, a String representing the character(s) on the key, the key's center x-coordinate, the key's center y-coordinate, and the key's target width (i.e., the minimum of its width and height).

B.2.2 `ModelDefinition`

This class encapsulates values required for a Fitts' law performance prediction model. Specifically, this class represents additional objects defining a model of text entry for processing by `TnWPM`. These values include a keypad digitization and a word-frequency-keystrokes representation of a corpus. All values can be loaded from a model definition file or set individually.

B.2.3 `TnKSPC`

This is one of TnToolkit's command-line tools. Given a letter-key mapping and a word-frequency representation of a corpus, this class calculates values for KSPC and KCME. It can also gather ambiguous word statistics, collate ambiguous word sets, and generate the word-frequency-keystrokes data required by `TnWPM`.

B.2.4 `TnMetric`

This interface outlines the methods required by all classes that calculate metrics within TnToolkit. Specifically, it ensures that the calling class can start, monitor, and stop the calculation of any metric.

B.2.5 `TnWPM`

This is one of TnToolkit's command-line tools. Given a model definition (as either a text file or an instance of `ModelDefninition`), this class calculates a WPM performance prediction based on Fitts' law. In addition, it can also collect performance and thumb usage statistics.

B.2.6 `WordFreqKs`

This class encapsulates a word, its frequency in the corpus and the keystrokes required to type it. It also implements a method to sort such objects in ascending order according to keystrokes. It then sorts objects with equal keystrokes in descending order of frequency, then in ascending order by word.

B.3 `tnt.gui.*`

This package provides classes that define and implement the characteristics of TnToolkit's GUI. To launch an instance of the GUI, run `tnt.gui.TnTApp`.

B.3.1 `MetricsCalculation`

This class calculates KSPC- and WPM-related metrics. However, three inner classes provide this functionality. First, an instance of `ProgressDialog` presents the user with a bar representing the progress of calculation and a "Cancel" button to halt further progress (Figure 4-4). Then, an instance of `CalculationThread` starts as a separate thread to

calculate the metrics asynchronously. Meanwhile, an instance of `ProgressThread` starts an asynchronous timer that periodically updates the progress bar.

The responsibilities of `MetricsCalculation` are to instantiate and initialize instances of the aforementioned inner classes, facilitate communication between those classes, and to provide access to the completed `TnKSPC` and `TnWPM` objects that store the metrics' results.

B.3.2 `HelpFrame`

This class encapsulates the mini-browser that displays and navigates TnToolkit's HTML-based help files (Figure 4-9). It provides, back, forward, and home navigation.

B.3.3 `KeyLetterDialog`

This class encapsulates the digitization dialog (Figure 4-2). It allows users to map letters to a key, but also restricts their input to valid ones.

B.3.4 `MetricsOutputDialog`

An instance of this class presents the values calculated by `TnKSPC` and `TnWPM` in tabbed panes (Figure 4-5 through Figure 4-8 inclusive). Each pane is an instance of `OutputPanel`, which facilitates the copying of its contents into other applications via the system's clipboard.

B.3.5 `ParametersDialog`

This class encapsulates the parameters dialog (Figure 4-3). It allows users to easily specify parameters for the various metrics.

B.3.6 `ScrollablePaintPanel`

This class encapsulates a scrollable panel that displays a background image and on top of which, allows the user to draw temporary rectangular outlines. These outlines appear during mouse drags and serve to illustrate the size and position of a key that is in the process of digitization. Once the user releases the mouse button, the outline disappears.

Instances of this class share a list model with the instance of `TnTFrame` that instantiated it. The elements of this list represent a digitized key. After drawing the background image of the keypad under examination, this class draws persistent outlines around each key in the list using the user-specified colour. If the user selected a digitized key, this class also draws move and resize handles around the appropriate outline. This process repeats during every redraw of the panel.

This panel captures all mouse actions and delegates responses to the appropriate method(s). If the user clicks on the panel, this class determines if it occurred within the outline of a digitized key. If so, it signals that the user selected that key, and updates itself and the list accordingly. This panel also updates itself to reflect changes to the list.

B.3.7 `TnTApp`

This class initializes the GUI and displays it in the center of the screen. The actual GUI is an instance of `TnTImp`.

B.3.8 `TnTFrame`

This abstract class instantiates and arranges the component of the GUI, but defines very little of its associated business logic. This separation of responsibilities assists code maintenance and facilitates future development.

B.3.9 `TnTImp`

This class implements the majority of the business logic related to the TnToolkit's GUI. This separation of responsibilities assists code maintenance and facilitates future development.

B.3.10 `Workspace`

Instances of this serializable class encapsulate a user's workspace (i.e., work-in-progress) to simplify the writing to and reading from persistent storage. Specifically, it stores the user's colour selection, the path to the keypad's image file, the digitized keys, and the current values in the parameters dialog.

Bibliography

BBC. (2006, 2006/06/09). *RSI danger from excessive texting*. BBC News. Available: http://news.bbc.co.uk/go/pr/fr/-/1/hi/health/5063364.stm.

BlackBerry. *SureType*. Available: http://www.blackberry.com/products/suretype/.

BNC. *British National Corpus*. Available: http://www.natcorp.ox.ac.uk/.

Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM, 23*(7), 396-410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Cellulist. *The Power of SMS*. Available: http://www.cellulist.com/text-messages/power-of-sms/.

Eatoni. *EQx*. Available: http://wiki.eatoni.com/wiki/index.php/EQx.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*(47), 381-391.

Gong, J., & Tarasewich, P. (2005). Alphabetically constrained keypad designs for text entry on mobile devices. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Portland, Oregon, USA: ACM Press. 211-220.

Green, N., Kruger, J., Faldu, C., & Amant, R. S. (2004). A reduced QWERTY keyboard for mobile text entry. *CHI '04 extended abstracts on Human factors in computing systems*, Vienna, Austria: ACM Press. 1429-1432.

Hick, W. E. (1952). On the rate of gain of information. *Quarterly Journal of Experimental Psychology*(4), 11-26.

Hissen, H. *Phone Key Pads*. Available: http://www.dialabc.com/motion/keypads.html.

How, Y., & Kan, M.-Y. (2005). Optimizing predictive text entry for short message service on mobile phones. *Proceedings of Human Computer Interfaces International (HCII 05)*, Las Vegas

Hwang, S., & Lee, G. (2005). Qwerty-like 3x4 keypad layouts for mobile phone. *CHI '05 extended abstracts on Human factors in computing systems*, Portland, OR, USA: ACM Press. 1479-1482.

Hyman, R. (1953). Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*(45), 188-196.

Isokoski, P., & MacKenzie, I. S. (2003). Combined model for text entry rate development. *CHI '03 extended abstracts on Human factors in computing systems*, Ft. Lauderdale, Florida, USA: ACM Press. 752-753.

James, C. L., & Reischel, K. M. (2001). Text input for mobile devices: comparing model prediction to actual performance. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Seattle, Washington, United States: ACM Press. 365-371.

Koester, H. H., & Levine, S. P. (1994). Modeling the speed of text entry with a word prediction interface. *IEEE Transactions on Rehabilitation Engineering, 2*(3), 177-187.

Kuhn, M., & Garbe, J. (2001). Predictive and highly ambiguous typing for a severely speech and motion impaired user. *Proceedings of the Conference on Universal Access in Human-Computer Interaction -- UAHCI 2001*, Mahwah (NJ): Lawrence Erlbaum Associates. 933-937.

Lesher, G., W., Moulton, B. J., & Higginbotham, D. J. (1998). Optimal character arrangements for ambiguous keyboards. *IEEE Transactions on Rehabilitation Engineering, 6*(4), 415 - 423.

MacKenzie, I. S. (1989). A note on the information-theoretic basis for Fitts' law. *Journal of Motor Behavior*(21), 323-330.

MacKenzie, I. S. (1992). Movement time prediction in human-computer interfaces. *Proceedings of the conference on Graphics interface '92*, Vancouver, British Columbia, Canada: Morgan Kaufmann Publishers Inc. 140-150.

MacKenzie, I. S. (2002). KSPC (Keystrokes per Character) as a Characteristic of Text Entry Techniques. *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*: Springer-Verlag. 195-210.

MacKenzie, I. S. (2003). Motor Behaviour Models for Human-Computer Interaction. J. M. Carroll (Ed.), *Toward a Multidisciplinary Science of Human-Computer Interaction* (pp. 27 - 54). San Franciso: Morgan Kaufmann.

MacKenzie, I. S., Kober, H., Smith, D., Jones, T., & Skepner, E. (2001). LetterWise: prefix-based disambiguation for mobile text input. *Proceedings of the 14th annual ACM symposium on User interface software and technology*, Orlando, Florida: ACM Press. 111-120.

MacKenzie, I. S., & Soukoreff, R. W. (2002a). A Model of Two-Thumb Text Entry. In *Proceedings of Graphics Interface 2002*, Toronto, Canada: Canadian Information Processing Society. 117 - 124.

MacKenzie, I. S., & Soukoreff, R. W. (2002b). Text Entry for Mobile Computing: Models and Methods, Theory and Practice. In I. S. MacKenzie (Ed.), *Human-Computer Interaction* (Vol. 17, pp. 147 - 198).

MacKenzie, I. S., & Soukoreff, R. W. (2003). Phrase sets for evaluating text entry techniques. *CHI '03 extended abstracts on Human factors in computing systems*, Ft. Lauderdale, Florida, USA: ACM Press. 754-755.

MacKenzie, I. S., & Tanaka-Ishii, K. (in press). Text entry with a small number of buttons. In I. S. MacKenzie & K. Tanaka-Ishii (Eds.), *Text entry systems: Mobility, accessibility, universality*. San Francisco, CA: Morgan Kaufmann.

MacKenzie, I. S., & Zhang, S. X. (2001). An empirical investigation of the novice experience with soft keyboards. *Behaviour & Information Technology*(20), 411-418.

Pavlovych, A., & Stuerzlinger, W. (2004). Model for non-expert text entry speed on 12-button phone keypads. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Vienna, Austria: ACM Press. 351-358.

Sandnes, F. E. (2005). Evaluating mobile text entry strategies with finite state automata. *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, Salzburg, Austria: ACM Press. 115-121.

Shirer, M., Baker, S., & Llamas, R. (2006, April 20, 2006). *Worldwide Mobile Phone Market Exhibits Strong Year-Over-Year Growth on Continued Strength of Developing Markets, IDC Finds*, [Press Release]. IDC. Available: http://www.idc.com/getdoc.jsp?containerId=pr2006_04_19_142525.

Shirer, M., & Llamas, R. (2006). *Worldwide Handheld Device Market Starts 2006 with Continued Decline in Shipments, According to IDC*. IDC. Available: http://www.idc.com/getdoc.jsp?containerId=prUS20145306.

Silfverberg, M., MacKenzie, I. S., & Korhonen, P. (2000). Predicting text entry speed on mobile phones. *Proceedings of the SIGCHI conference on Human factors in computing systems*, The Hague, The Netherlands: ACM Press. 9-16.

Soukoreff, R. W., & MacKenzie, I. S. (1995). Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard. *Behaviour & Information Technology*(14), 370-379.

Soukoreff, R. W., & MacKenzie, I. S. (2003). Input-based language modelling in the design of high performance text input techniques. *Proceedings of Graphics*

Interface 2003, Toronto, Canada: Canadian Information Processing Society. 89-96.

T9. *T9 Text Input Home Page*. Available: http://www.t9.com/t9_learnhow.html.

Tanaka-Ishii, K., Inutsuka, Y., & Takeichi, M. (2002). Entering text with a four-button device. *Proceedings of the 19th international conference on Computational linguistics - Volume 1*, Taipei, Taiwan: Association for Computational Linguistics. 1-7.

Tollinger, I., Lewis, R. L., McCurdy, M., Tollinger, P., Vera, A., Howes, A., & Pelton, L. (2005). Supporting efficient development of cognitive models at multiple skill levels: exploring recent advances in constraint-based modeling. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Portland, Oregon, USA: ACM Press. 411-420.

Yamada, H. (1980). A Historical Study of Typewriters and Typing Methods: from the Position of Planning Japanese Parallels. *Journal of Information Processing, 2*(4), 175-202.