Master's Thesis

Design and Development of a Mobile User Interface for Adaptive Handwriting Recognition to Improve Data Input into a Medical Information System

Martin Schlögl, Bakk.rer.soc.oec.

Institute for Information Systems and Computer Media (IICM)
Graz University of Technology

FERK SYSTEMS

Supervisor:
Assoc. Prof. Andreas Holzinger, PhD, MSc, MPh, BEng, CEng, DipEd, MBCS

Graz, May 2010

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

Entwurf und Entwicklung eines mobilen User Interfaces für adaptive Handschrifterkennung zur Verbesserung der Datenerfassung medizinischer Informationssysteme

Martin Schlögl, Bakk.rer.soc.oec.

Institut für Informationssysteme und Computer Medien (IICM)
Technische Universität Graz

FERK SYSTEMS

Betreuer:
Univ.-Doz. Ing. Mag. Mag. Dr. Andreas Holzinger

Graz, Mai 2010

Abstract

Rapid and accurate collection of data is very important in cases of emergency. Therefore, mobile data acquisition is a key issue for acceptance of a mobile information system. Handwriting recognition for acquiring data shows some problems on recognition accuracy and also on usability.

In this thesis, we focus on handwriting recognition on mobile devices for data acquisition in the context of health care, specifically done by paramedics in ambulance cars. The aim of this thesis was to increase the overall performance of handwriting recognition, focusing on recognition accuracy, speed, error correction and usability of the user interface by using solid usability engineering methods.

Based on real-life experiences, the thesis reports improvements on accuracy and usability of handwriting recognition archived by adaptation the results of handwriting recognition engine.

Despite these results, the end user studies showed that the virtual keyboard is still the preferred method compared to handwriting recognition for data acquisition on a PDA. This is possibly due to the wide availability of the QUERTY/QUERTZ keyboard.

Interestingly, participants with a computer usage of more than 30 hours a week prefer the virtual keyboard significantly more than the other participants.

Keywords

handwriting recognition, mobile device, human-computer interaction, usability, real-life, health care

This page is intentionally left blank

Kurzfassung

Schnelles und genaues Sammeln von Daten bei Notfällen ist sehr wichtig. Dadurch ist mobile Datenerfassung ein Hauptthema für die Akzeptanz eines Mobilen Information Systems. Handschrifterkennung zeigt hier einige Probleme in der Übersetzungsgenauigkeit und in der Bedienbarkeit.

Der Fokus dieser Arbeit liegt auf Handschrifterkennung auf mobilen Geräten zur Datenerfassung im Berech des Gesundheitswesens, speziell im Rettungswesen. Das Ziel dieser Arbeit war die Verbesserung der Gesamtperformance der Handschrifterkennung durch Verbesserung der Erkennungsrate, der Eingabegeschwindigkeit, der Fehlerkorrektur und der Verbesserung der Bedienbarkeit der Benutzerschnittstelle durch solide usability-engineering Methoden.

Die Arbeit liefert Verbesserungen der Erkennungsrate und der Benutzbarkeit des Systems durch Anpassung der Resultate, welche die Handschrifterkennung liefert, basierend auf Erfahrung aus der echten Anwendung des Systems.

Trotzdem zeigt die Endbenutzerstudie, dass die virtuelle Tastatur gegenüber der Handschrifterkennung bevorzugt wird für Datenerfassung auf einen PDA. Dies kann möglch sein durch die Weiterverbreitung des QWERTZ Tastatur Layouts.

Interessanterweise bevorzugen Teilnehmer, welche Computer mehr als 30 Stunden pro Woche verwenden, die virtuelle Tastatur signifikant eher als die anderen Teilnehmer.

Schlüsselwörter

Handschrifterkennung, Mobile Computer, Mensch-Maschine Kommunikation, Benutzerfreundlich, Gesundheitswesen

This page is intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted, either literally or by content, from the used sources.

Graz, 12th May 2010 ……………………………………………

Firstname, Surname

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, 12.Mai 2010 ……………………………………………

Vorname, Zuname

This page is intentionally left blank

Acknowledgments

First and foremost I want to thank my family, my grandmother, my parents and my brother for their support throughout my studies.

Especially I want to thank my adviser Andreas Holzinger for his professional commitment and support. I also want to thank for his patience. Thanks to him, I got more insight into the interesting field of human-computer interaction.

Special thank to key initiator and co-operation partner FERK Systems.

Finally, I want to thank the participants of the experiment at Red Cross.

This page is intentionally left blank

Abbreviations

ANN	Artificial Neuronal Net
ASCII	American Standard for Code Information Interchange
CPM	Characters per Minute
CRT	Cathode Ray Tube
High-Fi Prototype	High-Fidelity Prototype
HMM	Hidden Markov Model
IDE	Integrated Development Environment
LCD	Liquid Crystal Display
LipiTK	Lipi Toolkit
Low-Fi Prototype	Low Fidelity Prototype
OCR	Optical Character Recognition
PCA	Principal Component Analysis
PDA	Personal Digital Assistant
SDK	Software Development Kit
TBS	Time Between Strokes
WPM	Words per Minute

This page is intentionally left blank

Table of Contents

1. Introduction and Motivation for Research

In cases of emergency, rapid and accurate collection of the patients' data and documentation of the rescue operation is important. Promptly, accurately recorded and well communicated vital data of patients can make the difference between life and death (Holzman, 1999).
A possible solution can be a mobile application on a lightweight handheld device (Baumgart, 2005, Chittaro et al., 2007). Recording data to a network based information system in the field of work optimizes communication. For purposes of administration, the data is directly recorded to the information system, which also saves money.

FERK Systems implemented an information system called FEDAS RD for emergency services. There is also a mobile application, FEDAS.mobil RD included.
One developer tested the application in real-life and identified the data acquisition as the weak point of the system.

Emergencies are usually in difficult physical and mental situations; therefore, special attention to the design of information technology for emergencies has to be taken into consideration (Klann et al., 2008).
Data acquisition should have as little disruptive effect on the work of an emergency responder or paramedic as possible. Extensive text entry to mobile devices has to be avoided – less is more (Holzinger and Errath, 2007).

FERK System made the assumption, that people in health care rather prefer documenting medical data with handwriting. Therefore, handwriting recognition could be an important point for acceptance of a mobile information system.

A study shows that entering data into a mobile device via a stylus is slower, more erroneous and less satisfactory for end users than entering via a QWERTY layout keyboard (Haller et al., 2009). On the other hand, using a stylus is more accurate and faster than using a finger in touch based interfaces (Holzinger et al., 2008).

A specific study evaluating a PDA-Based Interface for Ambulance Run Reporting shows good results for acquiring data via virtual keyboard, while acquiring data with handwriting recognition showed some problems on recognition accuracy as well as usability problems (Chittaro et al., 2007).
Motivated by this work, the focus of this thesis is on improving the usability of data acquisition based on handwriting recognition. We defined our main goal in reaching the best possible user satisfaction, working on the following parts:

- Input speed
- Low error rate
- Easy correction of errors
- Simple to enter data

The beginning of the thesis, Chapter 2, shows theoretical background and related work relevant to the topic.
In Chapter 3, the development process of the handwriting recognition dialogs is described, including possible improvements of speed and accuracy of handwritten data acquisition.

Chapter 4 describes the experiment for the evaluation of the developed dialogs. This includes a description of the experiment application and the description of the experimental setting. The results of the experiment are shown in Chapter 5.

Finally, Chapter 6 to 8 concludes the thesis by discussing the results, giving a conclusion and an outlook for future work.

Chapter 9 describes the final implementation of the developed prototypes.

2. Theoretical Background and Related Work

2.1. Mobile Text Input

Mobile text input is divided into two main paradigms,

- Pen-based input (handwriting)
- Keyboard based input (typing)

While pen-based input produces handwritten characters which need to be recognized by handwriting recognition algorithm, keyboard based input is directly readable to a computer (ASCII characters).
(MacKenzie and Soukoreff, 2002)

Because of the limited capabilities of mobile devices, there has been a number of different text input techniques developed. Green defines tradeoffs between several issues for developing new text input innovations from the view of Hardware and Software development.

- Input Speed
- Accuracy
- Physical form factor
- Learning time
- Cost

(Green et al., 2004)

2.2. PDA – Personal Digital Assistant

A PDA is a mobile device with an interface that basically consists of a LCD touch screen with a resolution of 240 x 320 pixels. The interaction between the user and the system is mostly done with a stylus or a pen.
There are also buttons on the PDA, but the more comfortable way to interact with the system is via the touch screen.
Pascoe defined the most comfortable and secure method to hold a PDA (Figure 1).
(Pascoe et al., 2000)

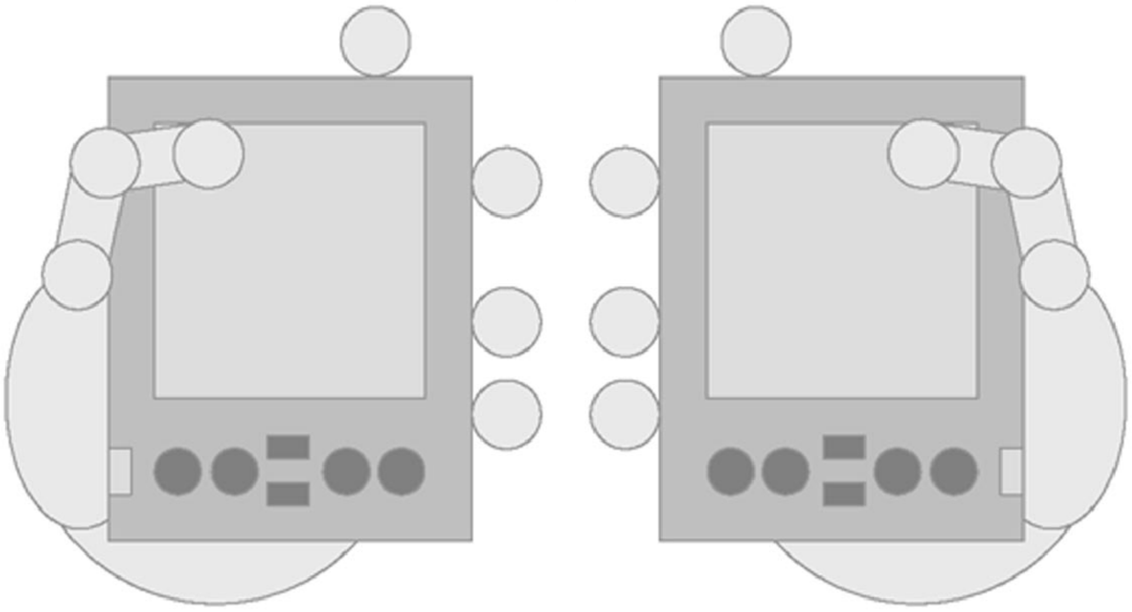


Figure 1: Most comfortable grip for one-handed operation (on the left, right-handed person; on the right, left-handed person) (Pascoe et al., 2000)

2.3. Mobile Interface Design

Due to smaller screens, different screen format compared to desktop screens (e.g. 4:3 on CRT monitor) and the changing environment where a mobile device is used, it is very important to design an interface of a mobile device in a different way than an interface on a desktop PC.

Using a mobile device whenever and wherever it is needed changes the requirements for the interface. While in an office, the light would be more or less the same, it is very variable for mobile devices because of mobility.

Also the user’s attention in an office might be higher than for example in a car.

(Chittaro, 2006)

Zwick, Schmitz and Kühl defined some basic design guidelines for small screens.

Touch screen

- Because there is no tactile feedback from the touch screen, the eyes and so the attention of the user has to remain on the screen.

Instant Feedback

- In order to get a feeling of directly manipulating and controlling the system, each interaction by the user should result in real time feedback.
- Delays between interaction and corresponding feedback interrupt the dialog between the user and the device. This can force the user to enter their input again.

Natural Mapping

- Functions like Save, Confirm or Next should be placed on the right side of the screen, while functions like Cancel or Back should be placed on the left.

Location on the screen

- Certain functions should have a fixed place within the whole application. This should help the user to develop a routine in the use of the system.

Text

- Basically, the text must be easily legible and therefore it has to be as clear, large and visible as possible.
- Italic text should be avoided because the slanted letters cause problems with the pixel grid of small screens.
- Bold text style is legible, if the letter spacing is sufficient.

Icons

- Icons allow fast non-verbal communication between the system and the user and are therefore very useful. To avoid confusion, icons should have the same meaning within the system.
- The icons' size depends on the form of interaction between the system and the user. E.g. tipping with stylus requires smaller icons while tipping with a finger requires bigger icons

(Zwick et al., 2005)

2.4. Resolving Recognition Based Input Problems

Schomaker defined different types of error:

- **Cancelled material:** User started writing a word, stops writing and starts scribbling over it
- **Discrete noise:** Dots caused by the unintended dropping of the pen on the touch-screen
- **Badly formed shapes**, which are illegible for humans and algorithms.
- **Legible by humans** but **not legible for algorithms.**
- **Correct words but unsolicited:** User writes a correct word, but was not prompted to do so.
- **Device generated errors:** E.g. Random noise

(Schomaker, 1994)

Recognition systems are in general prone to errors – so are handwriting recognition systems. Errors will occur, therefore, the system has to be designed to deal with errors. This is done by three means:

- Discovery
- Mediation
- Toolkit level support

(Mankoff et al., 2000)

2.4.1. Discovery

In this phase, the system should discover that the first choice of the list of possible results returned by the recognition engine is incorrect. This phase does not correct the result, but invokes some mediation techniques. (E.g. ambiguous choices in possible results)
(Mankoff et al., 2000)

2.4.2. Mediation

If the discovery process identified that the first choice is not the correct result, the mediation process selects the correct interpretation for the input. To do this, there are several strategies.

- Repetition
- Choice
- Automatic mediators
- Meta-mediation: deciding when and how to mediate
- The need of a toolkit support for mediation

Minimizing user interaction is worthwhile. Redemption, Choice and automatic mediators are the major strategies.
(Mankoff et al., 2000)

Repetition

Repetition is when the user has to do at least one more interaction over the normal input.
If the input is interpreted incorrectly, the user has to delete the wrong result and input it again until the result is correct. There are alternative input methods for these cases, too, for example with a virtual keyboard.
Also partial correction of some incorrect letters within a recognized word and undo of a wrong recognition belongs to repetition.
This technique requires effort by the user.
(Mankoff et al., 2000)

Choice

This technique gives the user a list of possible interpretations of the input. The user has to select the correct interpretation.
This technique also requires effort made by the user.
(Mankoff et al., 2000)

Automatic mediators

The best solutions are automatic mediators. In this case, the system automatically selects an interpretation of the user's input without requiring any interaction from the user. There are three classes of automatic mediators, thresh-holding, rules and historical statistics.

2.4.3. Toolkit level support

Systems benefit from toolkits which provide mediators. The toolkit does not only provide mediators, moreover, they get adapted. Processed mediators will be informed if their interpretation was accepted or rejected.
(Mankoff et al., 2000)

2.5. German language

The German written language consists of 26 basic characters, 3 Umlauts and one more special character, 10 numbers and punctuation marks. The 26 basic characters and 3 Umlauts can be written in upper and lower case.

- A…Z
- a…z
- ä, ö, ü
- Ä, Ö, Ü
- 0…9
- , . : ; ! ?

2.6. The QWERTY layout keyboard

The QWERTY keyboard was invented with touch-typing, mechanical machines, in 1874 (Kölsch and Turk, 2002).
Two years later, the first typewriter was published with the QWERTY layout keyboard. Later the "m" was placed on the position of the "ö" on a German keyboard or on the position of the ";" on an English keyboard (Noyes, 1998).
1905 the QWERTY layout keyboard was referred to as the standard and universal keyboard in an international meeting (Noyes, 1983).
1966, the QWERTY system became an international standard (Noyes, 1998).

The idea behind the layout was to place letters which frequently fell together in different quadrants to avoid mechanical problems where the type bars clashed when reverting to their resting position. The layout was intended for "hunt and peck" not for touch typing (Noyes, 1983).

2.7. The nature of handwriting

Handwriting is the result of a sequence of strokes, while one stroke is the drawing between the point from which the pen or stylus goes down to the point from which the pen or stylus goes up again. One character after another character is entered. Upper case letters consist of two strokes in average, lower case letters of one in average. (Tappert et al., 1990)

Handwriting is very individual to every person. Even for humans, handwritten characters are very hard to identify. Individual accuracy of nine humans identifying isolated hand written characters ranged from 94.9% to 96.5% in a study in 1960. (Neisser and Weene, 1960)

2.8. Handwriting Recognition

Réjean Plamondon, defines handwriting recognition as follows:

"*Handwriting recognition is the task of transforming a language represented in its spatial form of graphical marks into its symbolic representation.*"
(Plamondon and Srihari, 2000)

Basically, there are two ways to recognize handwriting recognition – Offline and Online Handwriting Recognition.
In Offline Handwriting Recognition only the completed writing is available. It belongs to Optical Character Recognition (OCR).
Online Handwriting Recognition is done while a user is writing. In this case, the order of the strokes as well as the order of the symbols drawn is available.
(Gader et al., 1997, Plamondon and Srihari, 2000, Strenge, 2005, Tappert et al., 1990)

Mobile Handwriting Recognition is realized with **Online Handwriting Recognition**.

2.8.1. Data Pre-Processing

The raw data of the user's handwriting which is gathered from the touch-screen gets preprocessed before the final recognition process tries to recognize the drawing.

Guerfali and Plamondon divide pre-processing into three groups:

- **Reduction of information:** The aim of this step is to reduce data for recognition, and therefore reduce time spent on recognition. This step consists of
  - **Filtering:** Removes duplicate data points; therefore the number of points is decreased.
  - **Dot reduction:** Reduces dots (e.g. dot in "i") to a single point
- **Elimination of imperfections:** Imperfections can occur with touch-screens because of hardware problems. The data points get corrected by
  - **Smoothing:** Average a point with previous neighbour points
  - **Wild point reduction:** This can eliminate occasional points produced by hardware.

    - **Hook removal:** Removing problem making hooks at the beginning and the end of strokes
    - **Component connection:** If there are unwanted pen lifts, the two strokes will be connected
- **Normalization:** Normalization reduces the effect of handwriting variations. This is done by
    - **Base drift correction:** Taking the writing to a horizontal level
    - **Normalization of size:** Resizes the writing to a standard size
    - **De-skewing:** Correction of the slant of words, cursive writing.

(Guerfali and Plamondon, 1993, Tappert et al., 1990)

Similar to Guerfali, Tappert divides pre-processing into

- Noise Reduction: (Figure 2)
    - Smoothing
    - Filtering
    - Wild point correction
    - De-hooking
    - Dot reduction
    - Stroke connection
- Normalization:
    - De-skewing
    - Baseline drift correction
    - Size normalization
    - Stroke length normalization

(Tappert et al., 1990)

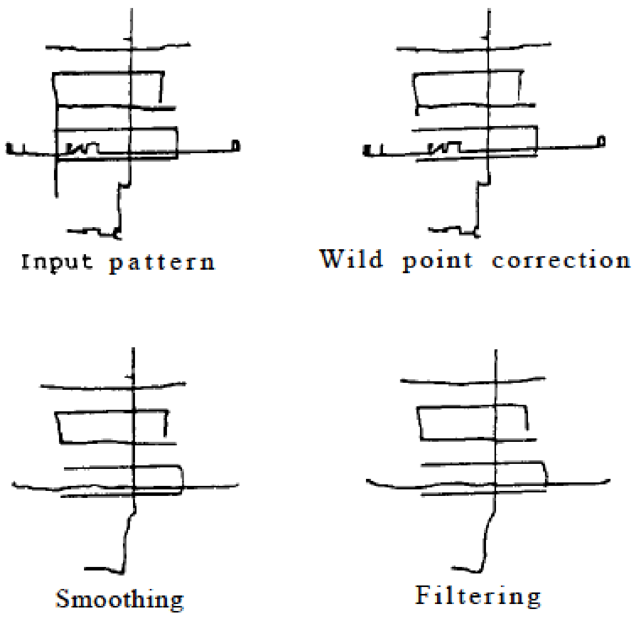


Figure 2: Example of pre-processing (Tappert et al., 1990)

Strenge defines the processes as follows:

- **Data Acquisition (Datenerfassung):** During input, samples are taken and stored in short intervals. These samples could consist of position of the stylus, the intensity of pressure on the touch-screen and the angle of the stylus to the surface of the touch-screen.
- **Preparation (Vorverarbeitung):** The collected data will be scaled into uniform size and errors will be removed before recognition. The most common methods are noise reduction (for example, if the user touches the touch-screen with a finger during writing with the stylus unintended), scaling (each letter is scaled to a defined size) and normalization (e.g. redundant points will be removed)
  At the end of preparation, the input is divided into segments. In the best case, one segment represents one letter (Figure 3). This figure also shows the problem with many possibilities for segmentation
- **Classification (Klassifizierung):** Recognition is performed in this stage. One or more segments are put together, building a class. The recognition engine then translates the classes to letters.
- **Composition (Komposition):** The recognized letters are put together to words and sentences.

(Strenge, 2005)

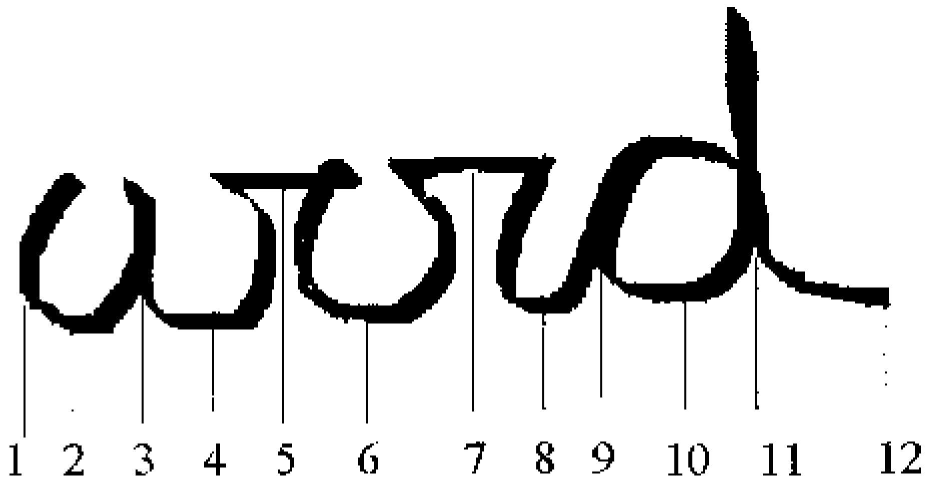


Figure 3: Example of segmentation (Plamondon and Srihari, 2000)

Segmentation can be done externally and internally. While internal segmentation is done by preprocessing, external segmentation is done by the user, by writing isolated characters. (Tappert et al., 1990)

The captured data will be classified as ASCII Symbols. There are several methods, which belong to two distinct families of classification:

- Structured and Rules-Based Methods
    - Fuzzy Rules
- Statistical Methods
    - Markov Modeling (Hidden Markov Modeling)
    - Based on Neural Networks

(Plamondon and Srihari, 2000), (Strenge, 2005)

2.8.2. Recognition based on Fuzzy Rules

(Holzinger et al., 2010)

Because of the fuzzy nature of human handwriting, it makes sense to adapt the well known fuzzy logic technique for this purpose (Gader et al., 1997).

Rather than strictly classifying data as in digital logic, fuzzy logic allows degrees of membership in multiple categories so that fuzzy rules may have a continuous, rather than stepwise, range of truth of possibility. Therefore non-identical handwritten numerals, from identical or different users, can be approximated using fuzzy logic for fast and robust handwriting recognition (Shi and Li, 2002).

2.8.3. Recognition based on Neural Networks

(Holzinger et al., 2010)
The methods based on Neural Networks were driven by the emergence of portable, pen based computers. A typical approach is to combine an artificial neural network (ANN), as a character classifier, with a context-driven search over segmentation and word recognition hypotheses (Yaeger et al., 1998).

2.8.4. Based on Markov Modeling

(Holzinger et al., 2010)
The methods using HMM (Marti and Bunke, 2002) are based on the arcs of skeleton graphs of the words to be recognized. An algorithm applied to the skeleton graph of a word extracts the edges in a particular order, which is transformed into a 10-dimensional feature vector. Each of these features represents information about the location of an edge relative to four reference lines, the curvature and the degree of the nodes incident to the considered edge.
Training of the HMM is done by use of the Baum-Welch algorithm, while the Viterbi algorithm is used for recognition (Bunke et al., 1995), (Xue and Govindaraju, 2006).

2.9. Metrics

The following two metrics define words per minute; the only difference is the starting metric.

Equation 1: Words per Minute by MacKenzie

$$wpm = \frac{cpm}{5}$$

wpm	Words per Minute
cpm	Characters per Minute, including punctuations and spaces

(MacKenzie et al., 1994)

Equation 2: Words per Minute by Lewis

$$wpm = \frac{cps * 60}{5}$$

wpm	Words per Minute
cps	Characters per Seconds, including punctuations and spaces

(Lewis, 1999)

Two standard measures for accuracy are Word Error Rate (WER) and Character Error Rate (CER). They are generated by two texts, the presented text and the transcribed text. In the "no error case", the two texts are the same. Each difference between the two texts is either an insertion (I), deletion (D) or a substitution (S).
(Read, 2007)

Equation 3: Word Error Rate (Read, 2007)

$$WER = \frac{(S + I + D)}{N}$$

N Total number of words

Equation 4: Character Error Rate (Read, 2007)

$$CER = \frac{(S + I + D)}{N}$$

N Total number of characters

2.10. Handwriting Recognition Acceptance (Accuracy)

The most important parameter for the acceptance for Handwriting Recognition is the accuracy.
People rate an accuracy of 90% to 95% to be a very poor recognition rate. Users will accept accuracy above 97% (3 wrong recognized out of 100 inputs). (LaLomia, 1994)

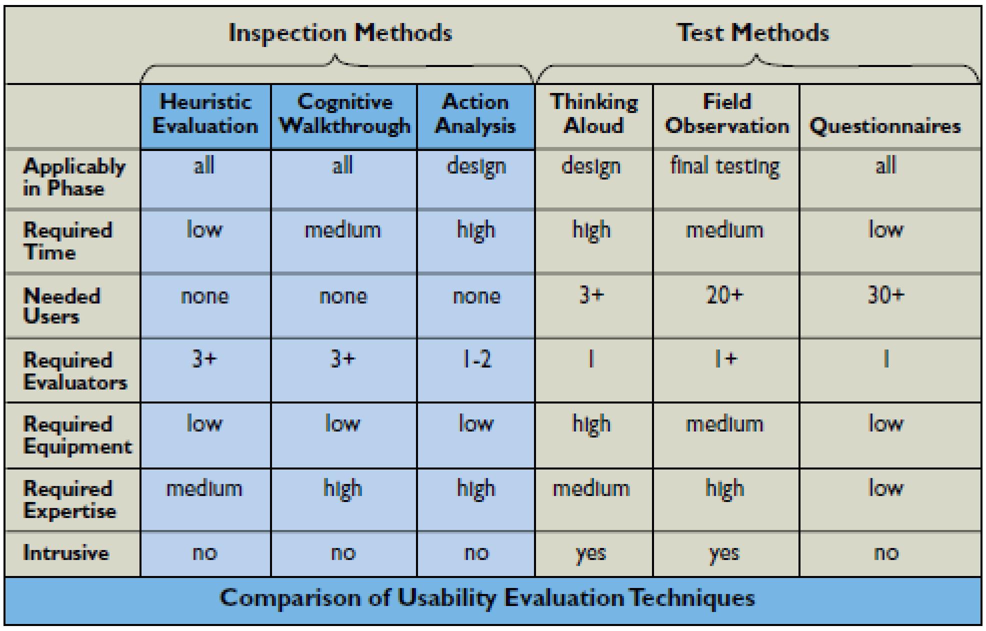
2.11. Usability Evaluation Techniques

There are two main categories for usability evaluation techniques. These techniques try to ensure that a prototype or implementation fits the defined usability requirements.

- Usability Inspection Methods
- Usability Test Methods

(Holzinger, 2005)

Figure 4 shows the detailed techniques of the two categories.



	Inspection Methods			Test Methods		
	Heuristic Evaluation	**Cognitive Walkthrough**	**Action Analysis**	**Thinking Aloud**	**Field Observation**	**Questionnaires**
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no

Comparison of Usability Evaluation Techniques

Figure 4: Usability Evaluation Techniques (Holzinger, 2005)

2.11.1. Cognitive Walkthrough

(Holzinger, 2005)

This technique is an inspection method and no end-user is urgently required. The team of analysts consists of software developers and usability specialists. Also end-users may be part of the analysts but this is not mandatory.

A full functional prototype and some tasks are required.

Each analyst goes through the system, simulating the tasks. The cognitive process of the user gets analyzed.

After simulating, the analysts discuss each task.

Because an end-user is not mandatory, this technique can be used at any point during the development process. A fully functional prototype gets analyzed, therefore, problems and errors in the interaction between user and the system can be found.

2.11.2. Field Observation

(Holzinger, 2005)
This technique is a test method and is carried out with end-users in real situations.
The user has to perform defined tasks at his work place. Observers take notes of the scenario in a non-intrusive way to enable the user to perform the tasks without diversions.
Statistical data should be logged by the system without disturbing the user.

2.12. Prototyping

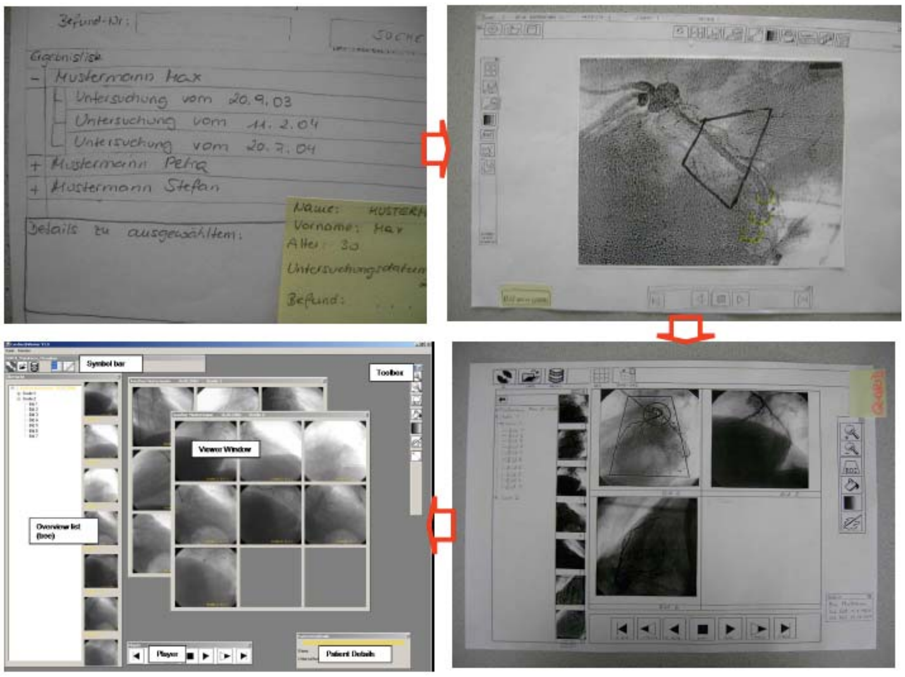


Figure 5: Example of Low-Fi – High-Fi Prototyping (Holzinger et al., 2005)

Prototypes are developed and created to evaluate the basic functionality of a system. In the special case of usability engineering, the basic functionality of the user interface is created and evaluated.
(Holzinger, 2004)

2.12.1. Low-Fidelity Prototyping

(Holzinger, 2004)
Low-Fi prototypes are very early prototypes of user interfaces. They are realized with paper mock-up.
With paper mock-up it is easy to create experimental drafts of interfaces on paper. These prototypes are user interfaces drawn on paper. They could be discussed and changed very simple.
The amount of time and resources to create such a prototype is very low – therefore, they are a very effective way of creating experimental interfaces.

Low-Fi prototypes are proper for Usability Inspection Methods (Chapter 2.11).

2.12.2. High-Fidelity Prototyping

(Holzinger, 2004)
High-Fidelity prototypes are implemented prototypes with full functionality. They are built in later stages in the development process when the development of the user interface is nearly finished.
Changes of these prototypes cost more time and money, because a programmer's expertise is needed to make changes.

High-Fi prototypes are proper for Usability Testing Methods (Chapter 2.11).

2.13. Spiral Model

The Spiral Model is an iterative development model. The Spiral Model (Boehm, 1986) was developed by Boehm (Boehm, 1986) based on experiences with refinements with the waterfall model (Royce, 1970).

The waterfall model is a non-iterative development model, while the spiral model is an iterative development model.

Basically, the development itself is the sum of iterations or also called circles. Each circle addresses the same steps. Each iteration addresses the same four steps, causing costs and increasing the progress of the whole project. (Figure 6)

- Determine objectives, alternatives and constraints
- Evaluate alternatives, identify and resolve risks
- Develop and verify next-level product
- Plan next phase

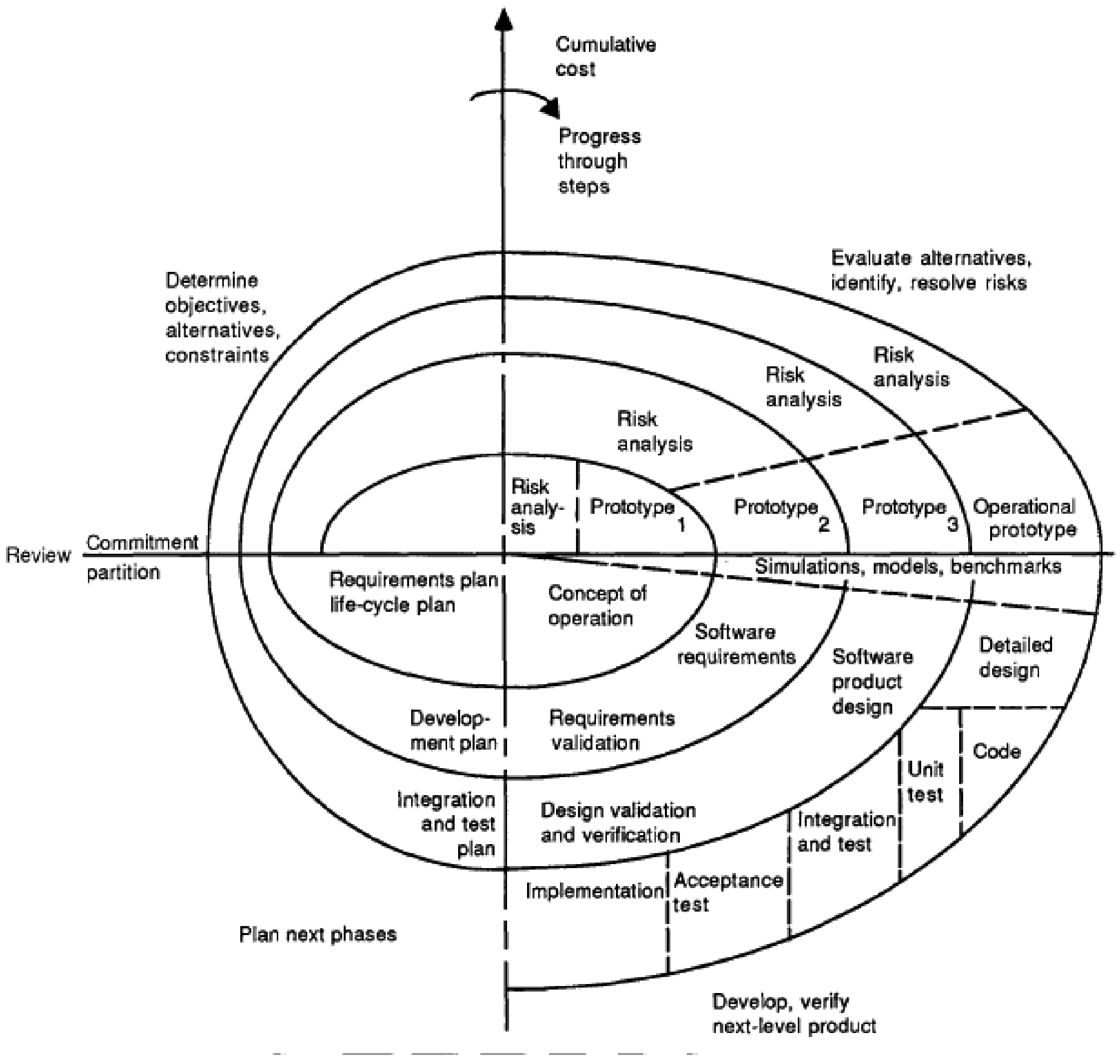


Figure 6: Spiral Model by Boehm (Boehm, 1986)

2.14. Development Process

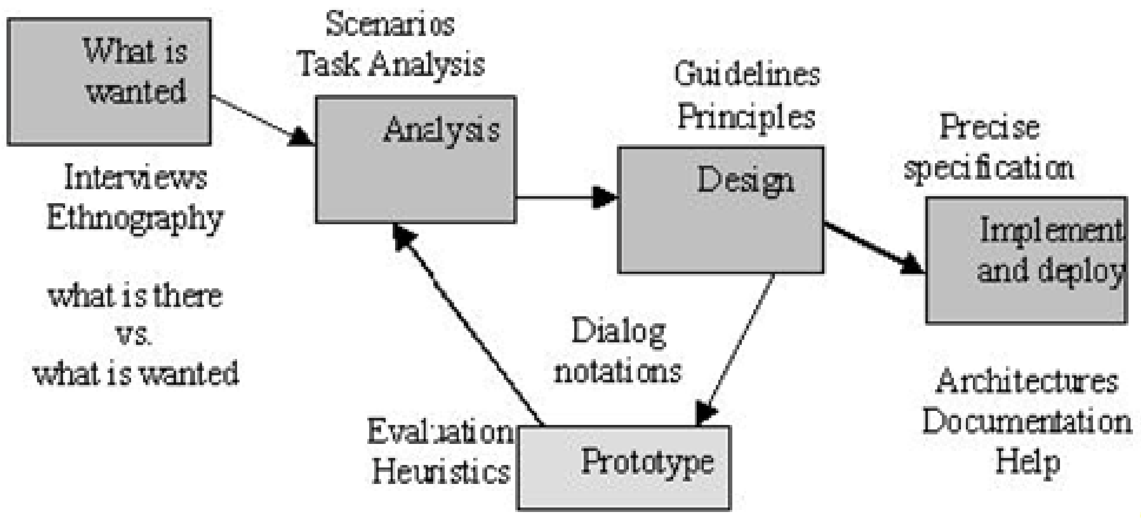


Figure 7: The System Lifecycle (Read et al., 2004)

The System Lifecycle (Figure 7) was used to design a novel writing interface for children. (Read et al., 2004)

2.15. Handwriting Recognition

2.15.1. Lipi Toolkit

(Madhvanath et al., 2007)
(Sourceforge)

Lipi Toolkit (LipiTK) is an open source, platform independent toolkit for online handwriting recognition. The core toolkit contains a generic algorithm for recognizing isolated handwritten shapes.
The toolkit consists of a core toolkit and an IDE.

The core toolkit contains a generic algorithm to recognize isolated handwritten shapes. These algorithms are generic pre-processing algorithms. A PCA (Subspace-based Classification) and a Nearest-Neighbour (Dynamic Time Warping) algorithm are used for recognition.

The IDE is a program, which helps to define/alter shapes.

Characters are defined as a number of classes. For each class, there are n shapes. Each shape is a possible handwritten representation of a class.

There are sets of shapes available for download on the homepage and with the LipiTK IDE it is possible to alter or add new shapes to the given shapes.

2.15.2. Decuma

Decuma is a powerful online handwriting recognition system by the ZI Corporation. This handwriting recognition is the default handwriting recognition system used by the Nintendo DS.

The handwriting recognition system only supports non-joint letter recognition, but this system is special not because of the recognition engine itself, but because of the innovative user interface

There is also a SDK for Decuma available, but this SDK is very expensive. After some email and phone discussions there were no indications for some discount.

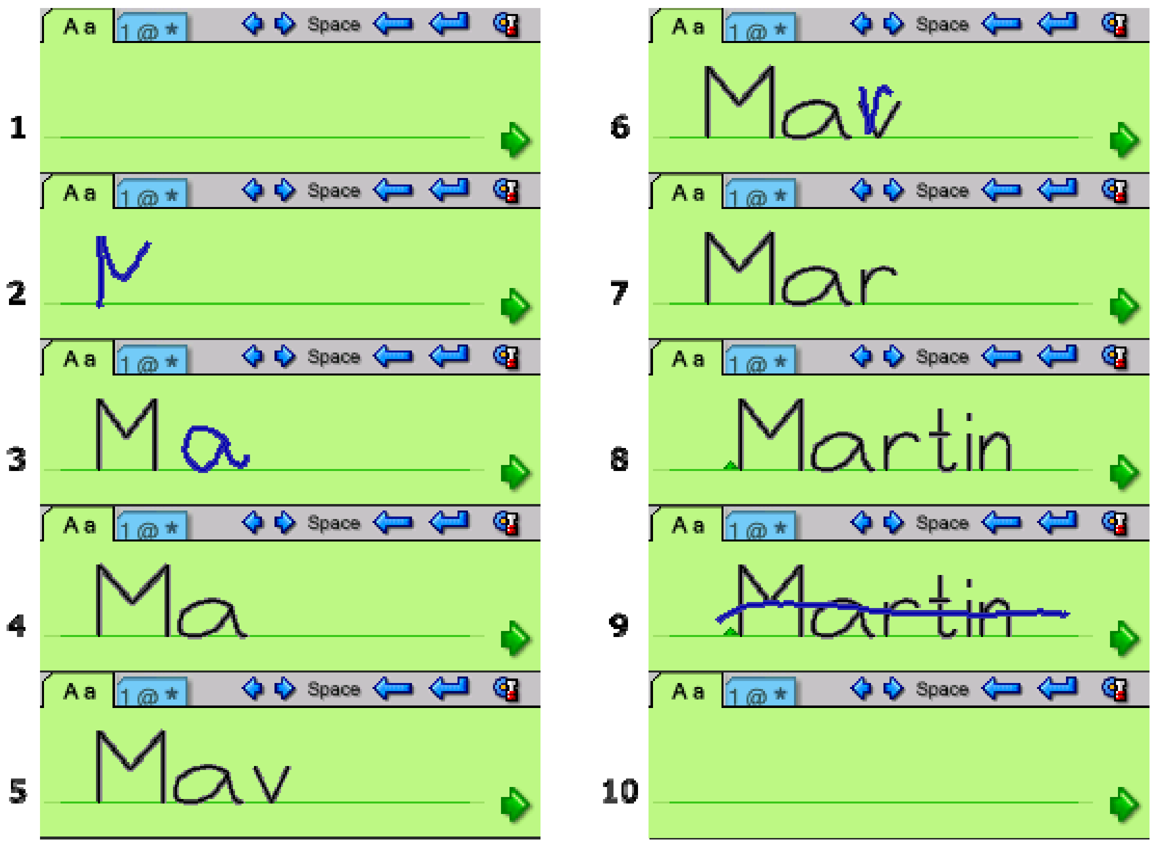


Figure 8: Demonstration of Decuma's user interface

2.15.2.1.User Interface

The user is writing letters, upper or lower case, on the green line. (Figure 8; 1)

Blue lines are strokes drawn by the user. (Figure 8; 3, 6, 9)

After finishing drawing, the stroke will be recognized. The result of the recognition will be displayed in black lines at the position of the drawing. (Figure 8; 9)

If there are wrongly inputted letters or wrongly recognized letters, the user can correct this by simply writing over the wrong letter. (Figure 8; 6)

Deleting wrong letters or words is done by drawing a line from right to left over the letter(s) which should be deleted. (Figure 8; 3, 4)

The special thing about this handwriting recognition is, as written above, the unique user interface, which gave us hints for developing ideas and features.
Also the fact that correcting single letters instead of whole words is much easier and more comfortable for the user makes this system appealing.

2.15.3. Calligrapher

Calligrapher is a powerful handwriting recognition system developed by Phatware. It recognizes

- joint words, non-joint letters
    - cursive style
    - printed letters (upper and lower case)
    - cursive and printed letters mixed

The recognition is based on dictionaries, but also other words could be recognized.
The basic recognition has implemented fuzzy logic and neural networks.
(Phatware, 2002)

Calligrapher's neural nets and the classification of Calligrapher are static and Calligrapher will therefore not adapt to recognition based on the users' handwriting. (Strenge, 2005)

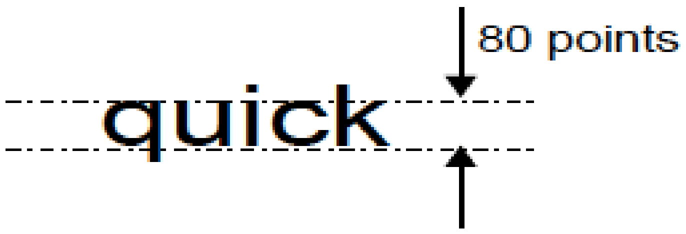


Figure 9: Best recognition results at letter size 80 points (Phatware, 2002)

There is a SDK for Calligrapher with an adequate price available. This SDK gives the opportunity for configuring Calligrapher, changing the timeout after which recognition starts and changing the results returned by the engine.

2.16. Improving recognition accuracy

One way to improve recognition accuracy is by decreasing the possible alternatives and therefore making discrimination easier. This could be done by designing interfaces where only restricted subsets will be accepted, for example, accepting only numbers in an input field (Frankish et al., 1995).
Also avoidance of high confusable pairs like "r" and "v", "O" and "0" and "I" and "l" would increase accuracy (Frankish et al., 1995).

In a study they formed the hypothesis that recognition accuracy should decrease if the size of the symbol sets increases. (Chang and MacKenzie, 1994)
If the symbol set contains only lower letters (26 letters), the recognition accuracy should be better than when using a symbol set of upper and lower case letters (52 letters).

They tested two handwriting recognizers, both with lowercase and with upper/lowercase

- Microsoft® character recognizer *Pen for Windows®*
- *CIC's Handwriter^{TM} 3.3*

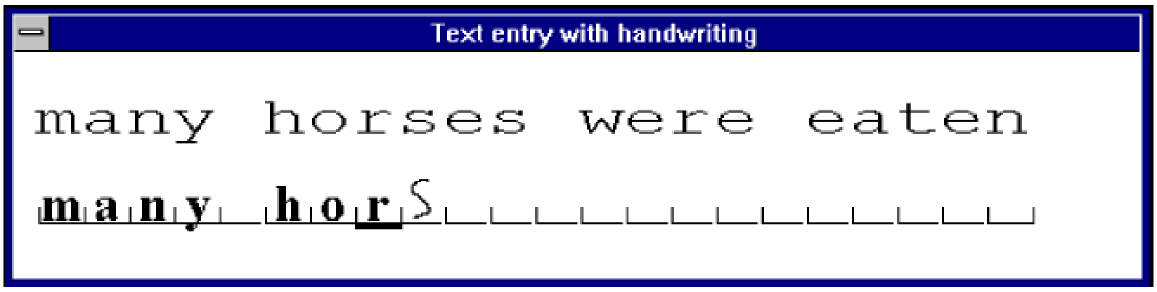


Figure 10: User interface of the study's software (Chang and MacKenzie, 1994)

To determine the recognition accuracy, the user interface was designed in a way that the text which should be entered is displayed on the screen and below the text each character is written in a separate section.

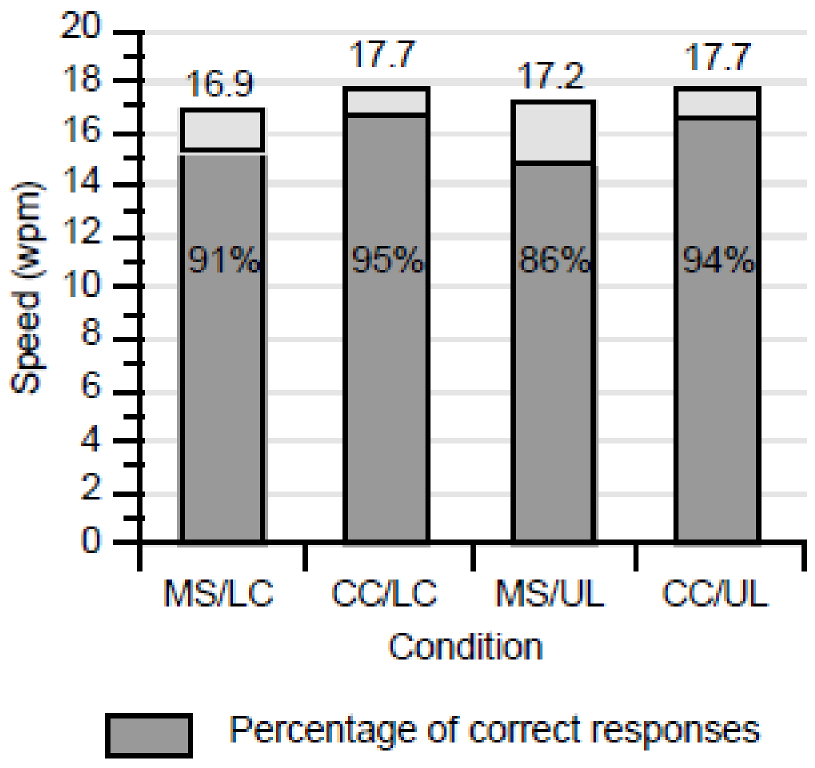


Figure 11: Comparison of the four conditions for entry speed and recognition accuracy (MS = Microsoft, CC = CIC, LC = Lowercase, UL = Upper+Lower) (Chang and MacKenzie, 1994)

The hypothesis was confirmed. Figure 11 shows that the recognition accuracy for Upper+Lower was lower than the accuracy for the smaller set of only lower case letters – while the writing speed (words per minute) shows hardly any difference.

2.17. PDA-Based Interface for Ambulance Run Reporting

(Chittaro et al., 2007)

A study motivated to improve effectiveness of rescue applications by creating an easy-to-use mobile system was done in 2007 by Chittaro in Italy. It focuses on design and evaluation of a mobile application, replacing the traditional ambulance run sheet which is filled out by first responders for each patient in the field of work or during transport.

2.17.1. Prototype

The traditional run sheet's size is DIN A4 (210mm x 297 mm). Because of the small size of the PDA's screen, the run sheet has to be divided into sub sections. With the navigation bar, it is possible to scroll to every section.

The input fields are classified into

- Numeric fields
    - Free numeric field
    - Pre-printed numeric fields
- Textual fields
- Multiple-choice fields
- Mutually-exclusive choice fields
- Graphical fields
- Mixed-type fields

2.17.2. Evaluation Procedure

Before starting, the user gets information and instructions about the usage of

- Touch screen and stylus
- Organization of the user interface
- Tips for entering data via handwriting recognition
- Usage of automatic word completion of virtual keyboard

During the tasks, the users were allowed to ask questions.

The users were asked to perform a task, entering data of a real rescue operation. They got a sheet of paper with a textual description and a photograph of an operation. The description of the operation contained all necessary data for filling out the run sheet on the PDA. This was neither an explicit text copy task nor a text creation task (MacKenzie and Soukoreff, 2002).

For text entry, handwriting recognition and input via virtual keyboard were used. For handwriting recognition, Phatware Calligrapher was used and configured for entering entire written Italian words. For input via virtual keyboard, the virtual keyboard of Windows Mobile with word completion was used.

After the task, a questionnaire about features liked or disliked had to be filled out by the user.

2.17.3. Evaluation Results

Table 1: Participants of the user evaluation

Number of participants	6	
Gender	4 male	2 female
Profession	First responder	
Age	23 – 50	
Average age	37.5	
Experiences with PDA	None	

Table 2: Evaluation results for handwriting recognition and virtual keyboard

	Median	**Variance**
Entering numeric values (virtual keyboard) (0=Hard, 9=Easy)	8.7	0.3
Writing text using the on-screen virtual (0=Hard, 9=Easy)	7.2	1.8
Writing text using handwriting recognition (0=Hard, 9=Easy)	3.8	6.6

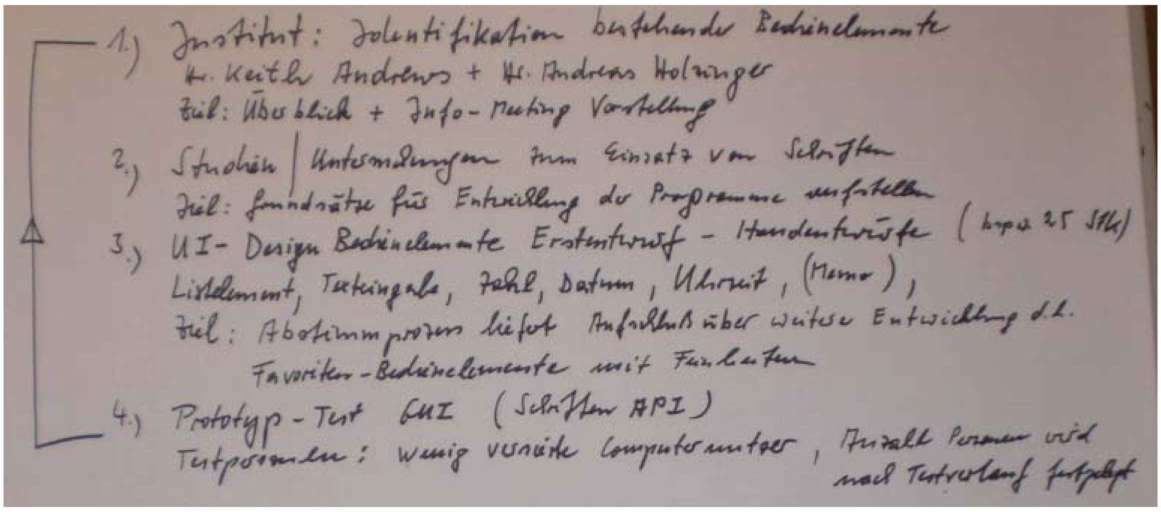
Entering data with handwriting recognition proved to be difficult for the users. Most of the entered data was not recognized correctly. The space on the screen for writing was too small. All in all, the handwriting recognition was not very satisfying and figured out a usability problem.

3. Materials and Methods

3.1. Development Process

An experimental incremental developing model was requested.

It was clear that the waterfall model (Royce, 1970) is not the right software development process for our project.
Prototyping is needed for our project, therefore, we decided to develop within iterations – analogue to the circles in the spiral model(Boehm, 1986).



1.) Institut: Identifikation bestehender Bedienelemente
Hr. Keith Andrews + Hr. Andreas Holzinger
Ziel: Überblick + Info-Meeting Vorstellung
2.) Studien | Untersuchungen zum Einsatz von Schriften
Ziel: Grundsätze für Entwicklung der Programme aufstellen
3.) UI-Design Bedienelemente Erstentwurf - Handentwürfe (bis zu 25 Stk)
Listelement, Texteingabe, Zahl, Datum, Uhrzeit, (Memo),
Ziel: Abstimmprozess liefert Aufschluß über weitere Entwicklung d.h.
Favoriten-Bedienelemente mit Funktion
4.) Prototyp - Test GUI (Schriften API)
Testpersonen: wenig versierte Computernutzer, Anzahl Personen wird nach Testverlauf festgelegt

Figure 12: Draft of a possible development process

We worked out a draft development process (Figure 12). This process included

- Identification of already completed work
- Analyse – Find basics for input components
- Prototyping (Lo-Fi and High-Fi Prototyping)
- Evaluation (Cognitive Walkthrough, Field Observation)
- Repeat points above until evaluation is satisfying

The System Life Cycle (Figure 7) which was used for the development of a handwriting recognition based writing interface for children is an example for a development process in a similar project. (Read et al., 2004)

Combined with our draft development process with the System Life Cycle (Figure 7), the key steps of the System Life Cycle are as follows:

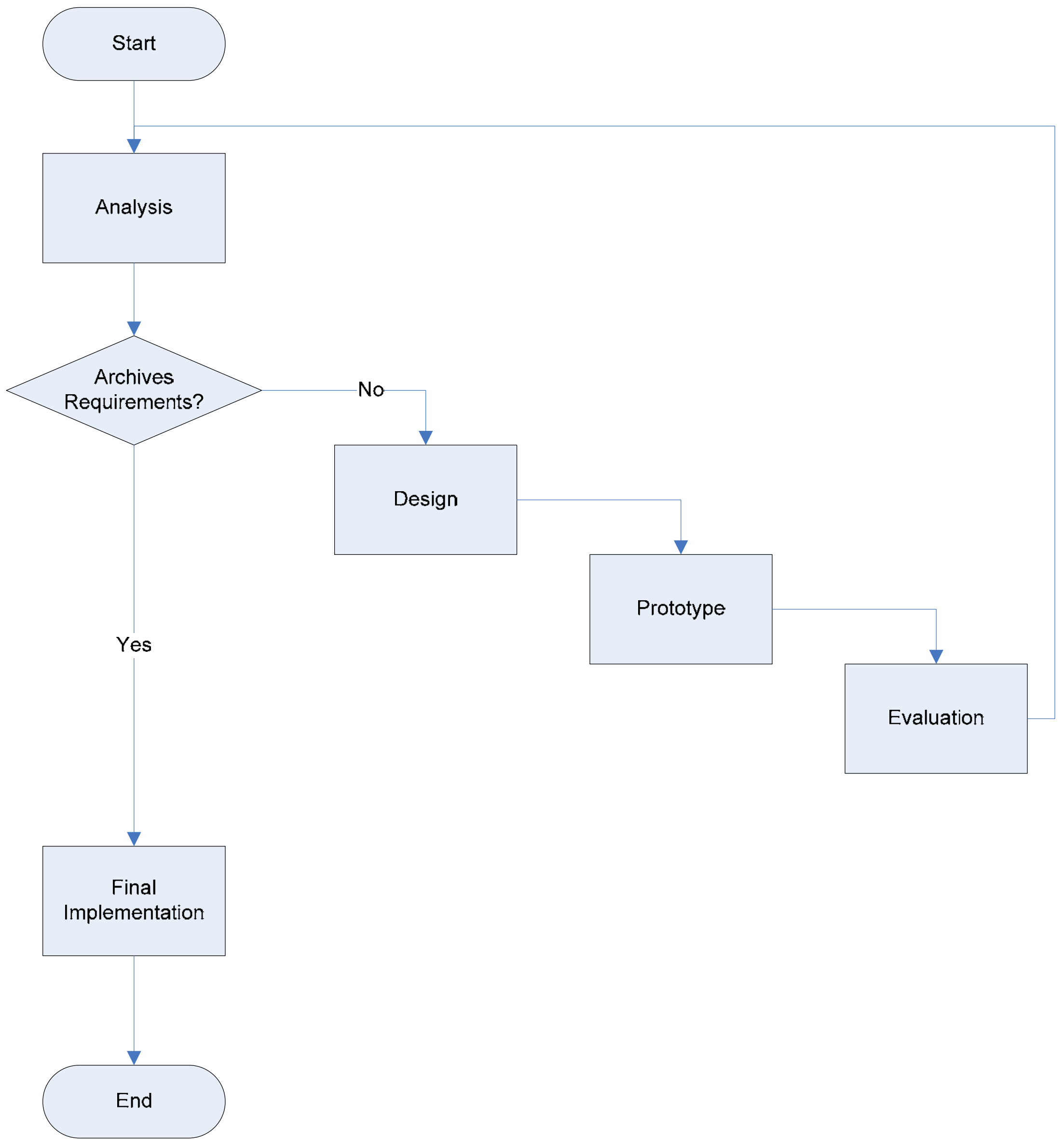


Figure 13: The development process for the project as a flow chart

Step	**Output**	**Participants**
Analysis	Issues for next iteration	(F),(H),(P),U,S
Design	Design of the prototype	U,S
Prototype	Finished Prototype	(U),S
Evaluation	Results of evaluation	(U),S
Final Implementation	Object orientated and Coding standard conform implementation	U,S

F… Mr. Ferk, U… Mr. Unger, S… Mr. Schlögl, H… Mr. Holzinger, P… Mr. Peischl

Table 3: The development process, output and participants of the steps

3.1.1. Analysis

The findings from previous circles (prototypes) are discussed and the issues for the next iteration are decided.
Also picking up new ideas and their possible implementation and improvements are discussed.

Normally, these discussions are done by Mr. Unger and Mr. Schlögl, but when discussing ideas, Mr. Ferk also joins the discussions.
There were also meetings with all participants to decide next key issues in the project.

If the analysis of the results of evaluation concludes that the functionality of the last prototype achieves the requirements, the next will be the final implementation.

3.2. Design

In this step, ideas and issues from the analysis are given a design for implementation.
This consists of the design of the user interface and basic design of the software.
The rough design of the user interface is done by drawing a draft version on paper and discussing the draft.

After this step the overall tasks for realizing the prototype are clear.

3.2.1. Prototype

3.2.1.1. Low-Fi Prototype

At the beginning, these prototypes were created on paper by hand and with Microsoft Visio (e.g. Figure 16). After the basic design of the interface was completed, the Low-Fi Prototypes were created by manipulating screen shots with graphic design software like GIMP (e.g. Figure 20).

3.2.1.2. High-Fi Prototype

Every time the basic design of the user interface changed and some approaches of the Low-Fi prototypes have to be evaluated in full functionality (with handwriting recognition), the High-Fi prototype was extended.

3.2.2. Evaluation

High-Fi Prototypes are evaluated in at least one of the following three different ways (2.11.1 Cognitive Walkthrough):

- Evaluation by the developers (Mr. Unger, Mr. Schlögl)
- Evaluation by colleagues and friends (for example colleagues at the office of FERK Systems)
- Evaluation in real situations (Austrian Red Cross)

To get a more independent evaluation, colleagues at the office of FERK Systems are asked to complete several tasks.

To get an independent evaluation, tasks are performed by people within the Health Care System (for example, colleagues).

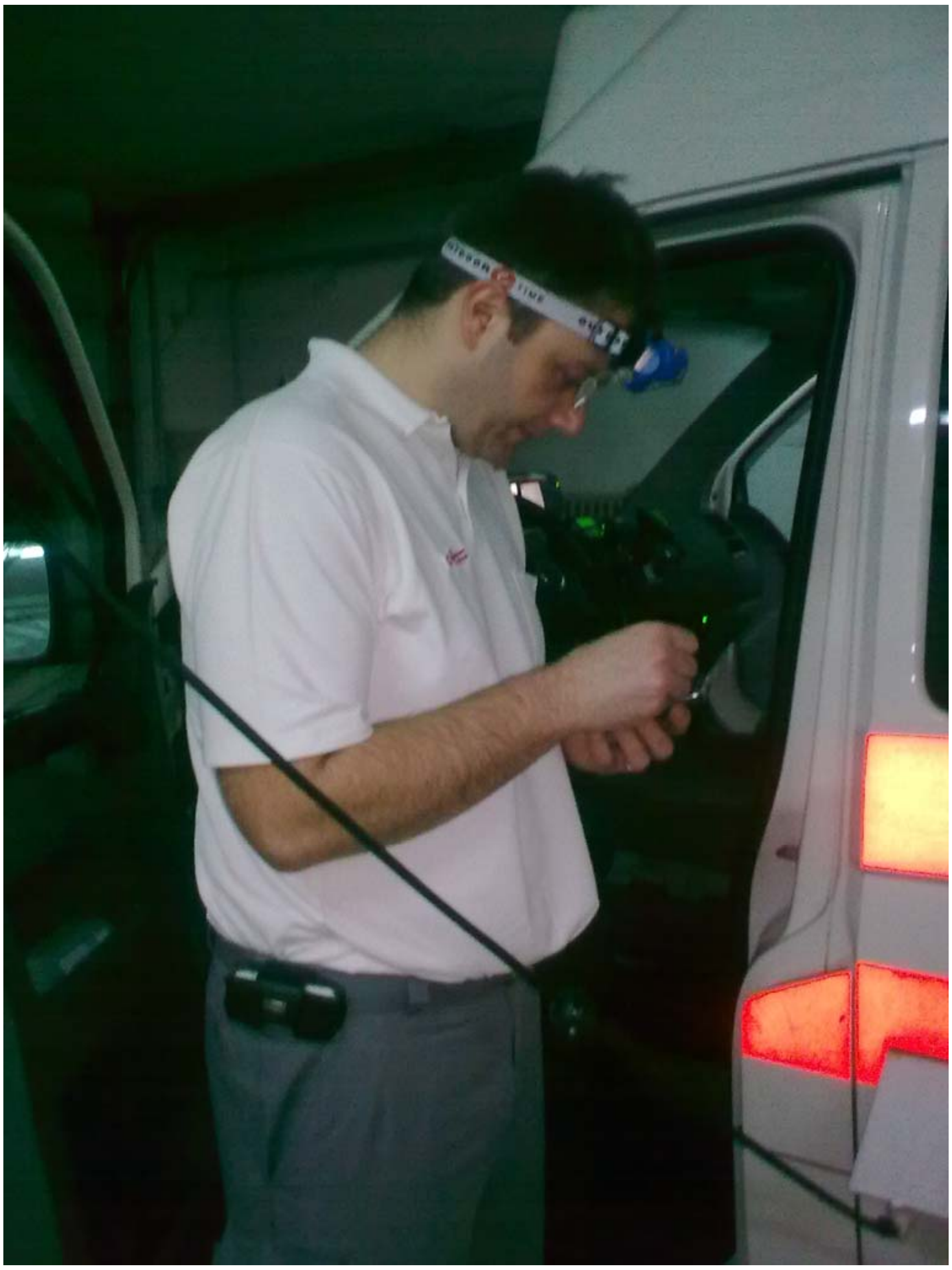


Figure 14: Evaluation as Cognitive Walkthrough in Real Life Situations

3.2.3. Final Implementation

This is the final step of the development process. Based on the findings of the prototypes, a final Object Orientated Design is created.

The final implementation will not be a further development on a prototype. The implementation has to start from the beginning and must be in FERK SYSTEMS coding standard (FERK-Systems, 2005, FERK-Systems, 2007).

3.3. Consistent Colour Scheme

A consistent design is a very important factor for user satisfaction. Therefore, a consistent colour scheme was defined.

There are three main action types in our input dialogs:

- Input actions
- Command actions
- Navigation actions

There is a main colour for every action type. A brighter version of this main colour is used for unused elements, a darker colour for actual used elements.

For the buttons, there are three states defined:

- **Enabled:** The functionality of the button is enabled.
- **InUse:** The button changes to this state if the button is pressed or the functionality of the button is active (e.g. Upper Case Button).
- **Disabled:** The functionality of the button is disabled

3.3.1. Input actions

Input actions are actions which change the result field. Green is the main colour for these actions.
All handwriting input areas are green, as well as buttons for "New Line", "Space" and "Backspace".

Bright version of green:	Red 99, Green 227, Blue 99	🟩
Dark version of green:	Red 57, Green 130, Blue 57	🟩

Table 4: Input action colours

Elements:

Button (Enabled, Focused, InUse, Disabled)	**Description**
⇦ ⇦ ⇦ ⇦	**Backspace** – Deletes the character to the left of the cursor position
↵ ↵ ↵ ↵	**New Line** – Inserts a new line at the current cursor position
␣ ␣ ␣ ␣	**Space** – Inserts a blank space at the current cursor position
🟩	Area for handwriting drawings

Table 5: Input action elements

3.3.2. Command actions

Then main colour for command buttons is blue. These are actions which change the dialog appearance, without changing data within the edit field or positioning the cursor.

Bright version of blue:	Red 148, Green 166, Blue 198	🟦
Dark version of blue:	Red 76, Green 85, Blue 102	⬛

Table 6: Command action colours

Elements:

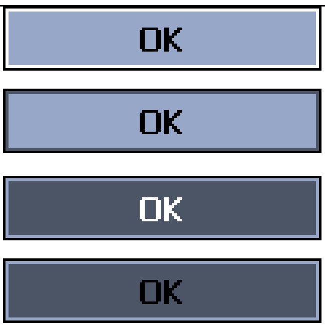
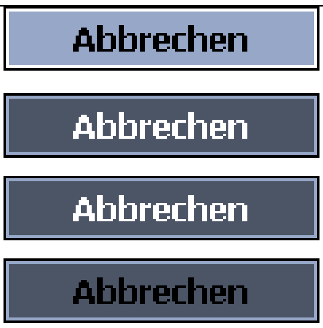
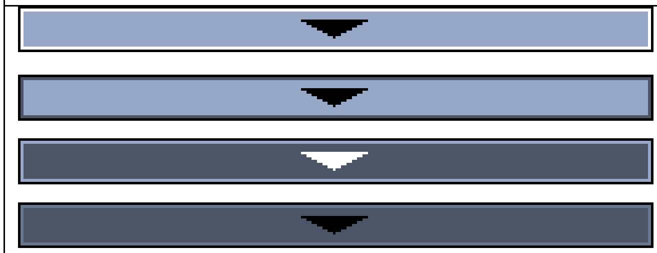
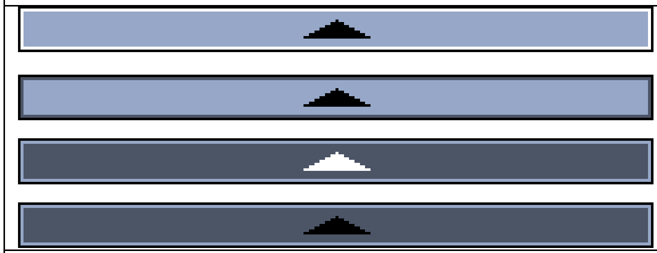
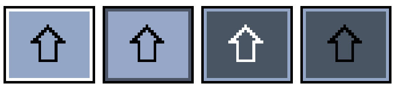
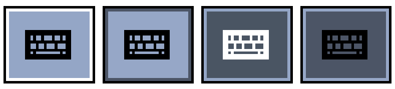
	OK Button – commits changes and closes input dialog
	Cancel Button – cancels changes and closes input dialog
	Unroll – Shows more lines of multi line text edit field, hides input components
	Enroll – Shows standard lines of multiline text edit field, shows input components
	Upper Case Button – Changes to Upper Case Mode
	Keyboard Button – Shows Virtual keyboard

Table 7: Command action elements

3.3.3. Navigation actions

These are actions, which move the cursor position. Their main colour is orange.

Bright version of orange:	Red 255, Green 182, Blue 115	🟧
Dark version of orange:	Red 191, Green 137, Blue 86	🟫

Table 8: Navigation action colours

Elements:

← ←	Left Button – Navigates the cursor position one to the left
→ →	Right Button – Navigates the cursor position one to the right

Table 9: Navigation action elements

In early prototypes these buttons were also included but it was decided to remove them because positioning the cursor in the text is not a common action compared to pressing space or deleting a letter.
Removing these buttons also decreased the number of buttons and therefore simplified the user interface.

Positioning the cursor in the text is realized by placing the cursor directly by using the stylus within the text.

3.4. “Keyboard” Input

The given virtual keyboard of Microsoft Windows Mobile is not very comfortable. It is small and hard to read.
Moreover it is not possible to filter the buttons, which are displayed on the keyboard.
Only displaying the letters that are actually needed would be a better solution, for example, inputting a name only requires letters from a to z and special German letters in upper and lower case, as well as the character “-“ for double names.

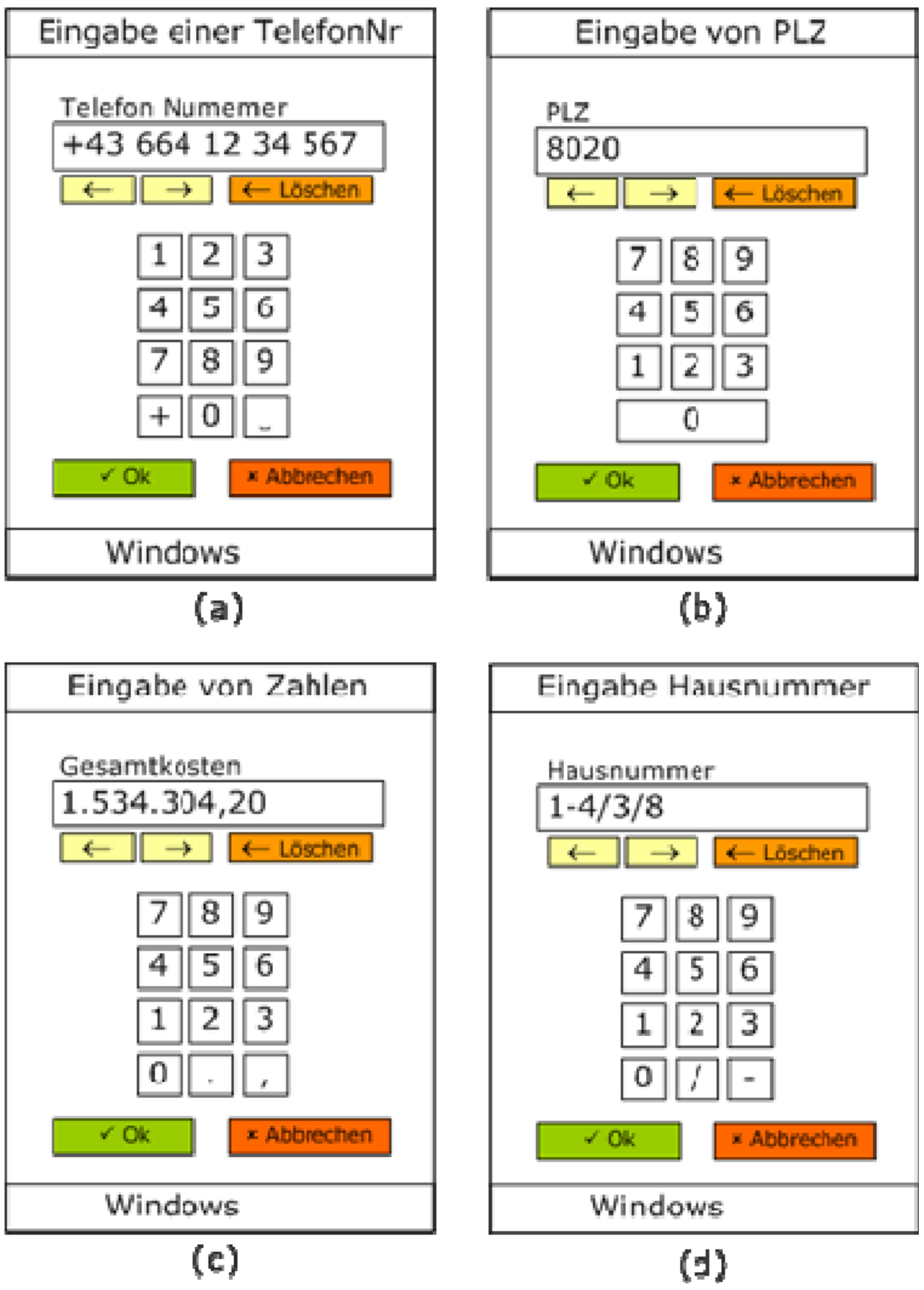
We defined some input dialogs for entering data

- numbers
- text

The main goal of these input dialogs is to show only the buttons on the keyboard which are needed, e.g. for typing in an amount of money, only the numbers 0…9, +, - and comma/dot are needed.

Figure 15 shows some of these prototypes, special for a few kinds of numbers.
Figure 16 shows a prototype to input a name or a street name. Instead of the confusing "Shift" button (for upper case letters), upper and lower case letters are displayed.

The prototypes on Figure 15 and Figure 16 are Low-Fi Prototypes, designed with Microsoft Visio, and never got implemented, because we focused on handwriting recognition.


Eingabe einer TelefonNr

Telefon Nummer: +43 664 12 34 567

Eingabe von PLZ

PLZ: 8020

Eingabe von Zahlen

Gesamtkosten: 1.534.304,20

Eingabe Hausnummer

Hausnummer: 1-4/3/8

Figure 15: (a) "keyboard-dialogs" for
(a) phone number, (b) zip code, (c) number, (d) house number

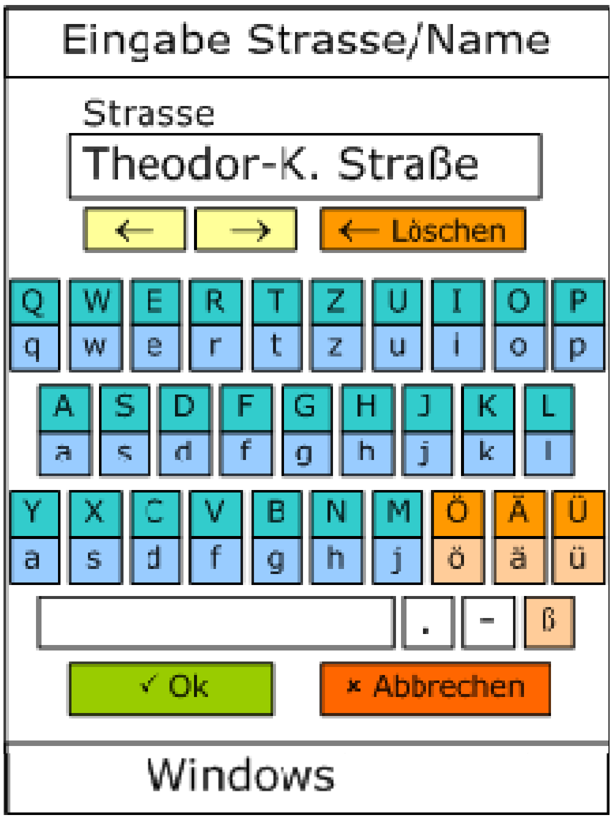
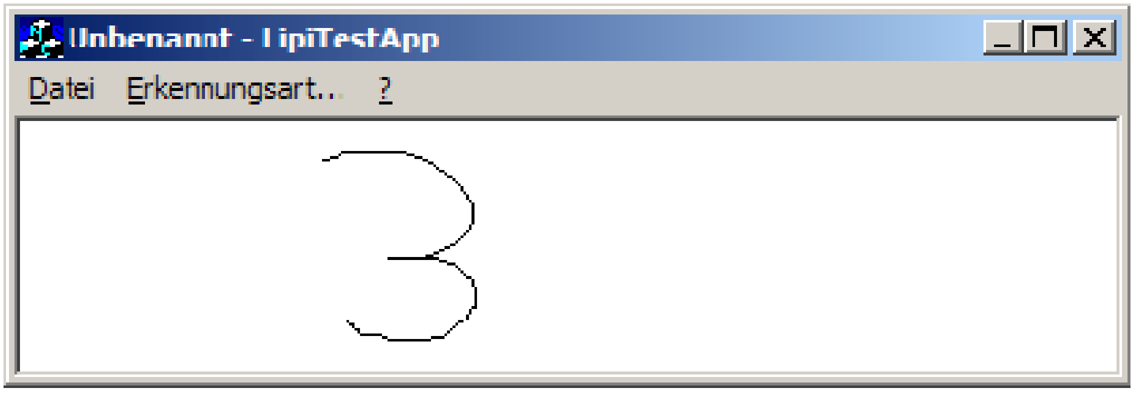


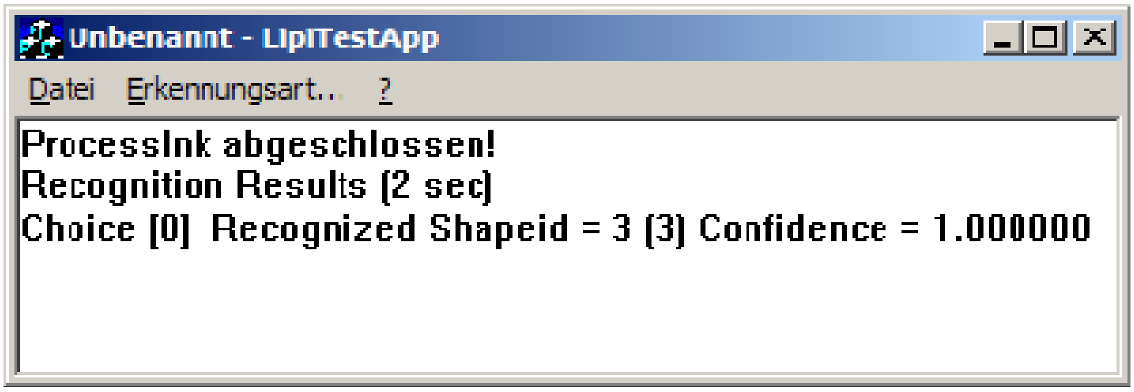
Figure 16: “keyboard-dialog“ for entering a street name or a name

3.5. Basic Handwriting Recognition Prototypes

3.5.1. Lipi Toolkit

We developed a small prototype in Microsoft Visual C (Figure 17). It was only for testing the performance of Lipi TK.





```
ProcessInk abgeschlossen!
Recognition Results (2 sec)
Choice [0]  Recognized Shapeid = 3 (3) Confidence = 1.000000
```

Figure 17: Simple prototype for testing Lipi Toolkit

This prototype had hardly any functionality; only numbers (one after another) could be recognized.

The prototype was tested on a notebook with an Intel Core Duo and 2.0 GHz – the performance was disastrous.
It took two seconds to recognize the "3" even though the set of shapes which could be recognized counted only ten shapes ("0".."9").
We also tried to recognize letters – on the same notebook it took seven seconds to recognize a letter.

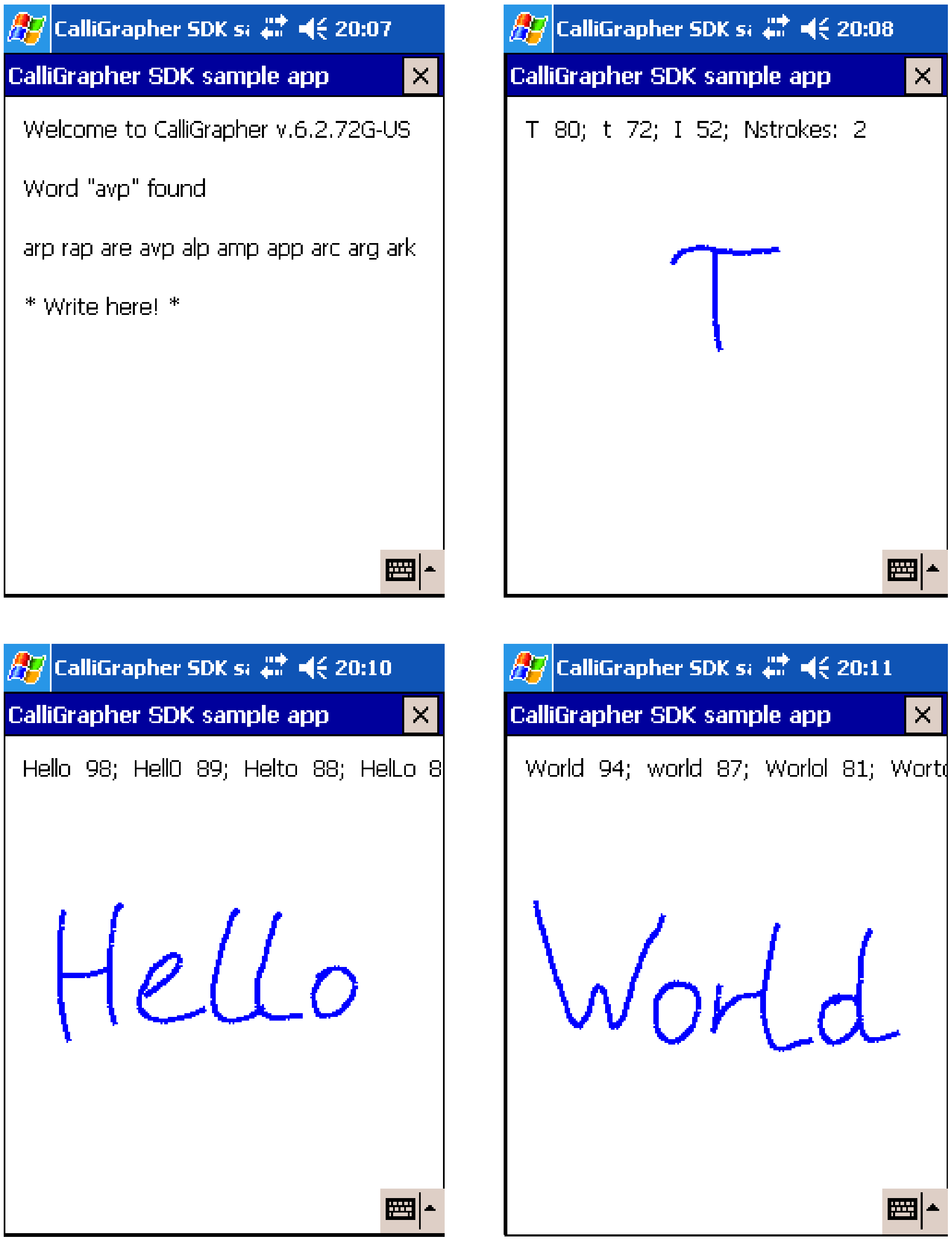
The CPU of a PDA is about 400MHz. Therefore, it is impossible to use Lipi TK for our input dialogs.

3.5.2. Calligrapher

We developed a simple Prototype (Figure 18) for typing in text with calligrapher. With this prototype, the user was able to input data by handwriting somewhere on the screen of the PDA.

An important parameter for Calligrapher SDK is the used timeout. This timeout means the elapsed time after the user stops drawing on the screen. After this timeout, the recognition starts.
This timeout is very important because a letter or a word consists of at least one stroke, but there is no maximum of strokes (e.g. letter “L” consists of one stroke, while “T” consists of two strokes).

The prototype shows that Calligrapher SDK returns a list of possible recognized letters or words with a probability. Also the number of strokes is provided.


CalliGrapher SDK sa 20:07
CalliGrapher SDK sample app
CalliGrapher SDK sa 20:08
CalliGrapher SDK sample app
CalliGrapher SDK sa 20:10
CalliGrapher SDK sample app
CalliGrapher SDK sa 20:11
CalliGrapher SDK sample app

```
Welcome to CalliGrapher v.6.2.72G-US

Word "avp" found

arp rap are avp alp amp app arc arg ark

* Write here! *
```

```
T 80; t 72; I 52; Nstrokes: 2
```

```
Hello 98; Hell0 89; Helto 88; HelLo 8
```

```
World 94; world 87; Worlol 81; Wort
```

Figure 18: Simple Calligrapher Prototype

Calligrapher has a good word recognition performance with the help of dictionaries but lacks the ability to enter words that are not included in the dictionary. This is especially problematic when trying to enter names. But if users input a word, which seems to be a word in the dictionary, Calligrapher will recognize this word, although the user wants another word to be recognized. Therefore, sometimes it is impossible to input correct words, especially names.

Moreover, if words are falsely corrected, it is difficult to implement a powerful user interface to correct the error, either the whole word will be deleted, requiring entering the word again or the wrong letters have to be corrected.

We tried to define separate dictionaries for each of our fields, which can be entered:

- Names, 12.415 common German names
- Surnames, 43.531 common Surnames

We also wanted to define dictionaries for cities and streets, but when we defined names and surnames, we saw that this will not solve the whole problem. Entering a name which is similar to a name in our dictionary caused Calligrapher to recognize the wrong name.

3.5.3. Decision for a Handwriting Recognition Engine

LipiTK would be the cheapest alternative, because it is free and open source, but the performance (recognition time) of the recognition is not very good.

The SDK of Decuma is very expensive, and there were no discounts, therefore it was out of discussion to choose Decuma.

The Calligrapher SDK is available for an adequate price. Moreover, the study by Chittaro (Chittaro et al., 2007) which motivated us for this work on handwriting recognition was also done with Calligrapher.

Therefore, Calligrapher SDK was chosen.

3.6. Guided Handwriting Input Dialog

The idea of the following input dialogs was to display separate handwriting input sections for inputting different data. The user is guided through the dialog by having separate sections for each part of time (SS:MM) and date (DD:MM:YYYY).

The user has to input the whole value for each section at once, e.g. to input the year 2009 the user has to write all four letters into the year section.

We thought of one big advantage designing such dialogs – it should be easy for the user to correct incorrectly recognized dates and times. Errors are simply corrected by re-writing the values in the corresponding section.

In the next step, we found a second big advantage of these separated sections for handwriting input. It is possible to reduce possibilities of valid values for each input section. This would give better recognition results (MacKenzie and Chang, 1999). We did these by using self-defined dictionaries which are loaded into Calligrapher. We defined special dictionaries for each of these separate input sections with the following contents:

- **Section Day:** “1” … “31”
- **Section Month:** “1” … “12”, Jänner, Jän; Februar, Feb; März; April; Mai; Juni; Juli; August, Aug; Oktober, Okt; November, Nov; Dezember, Dez
- **Section Year:** “1800” … “2100” (we defined this as valid years)
- **Section Hours:** “0” … “23”
- **Section Minutes:** “0” … “60”

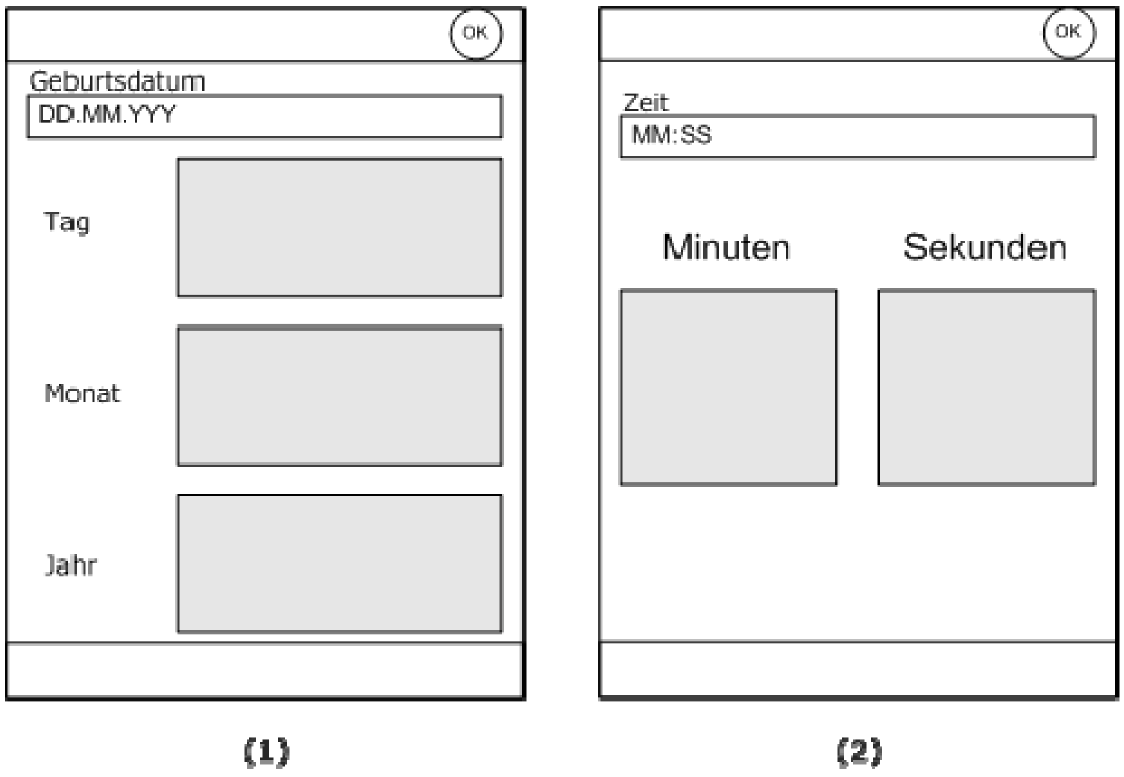
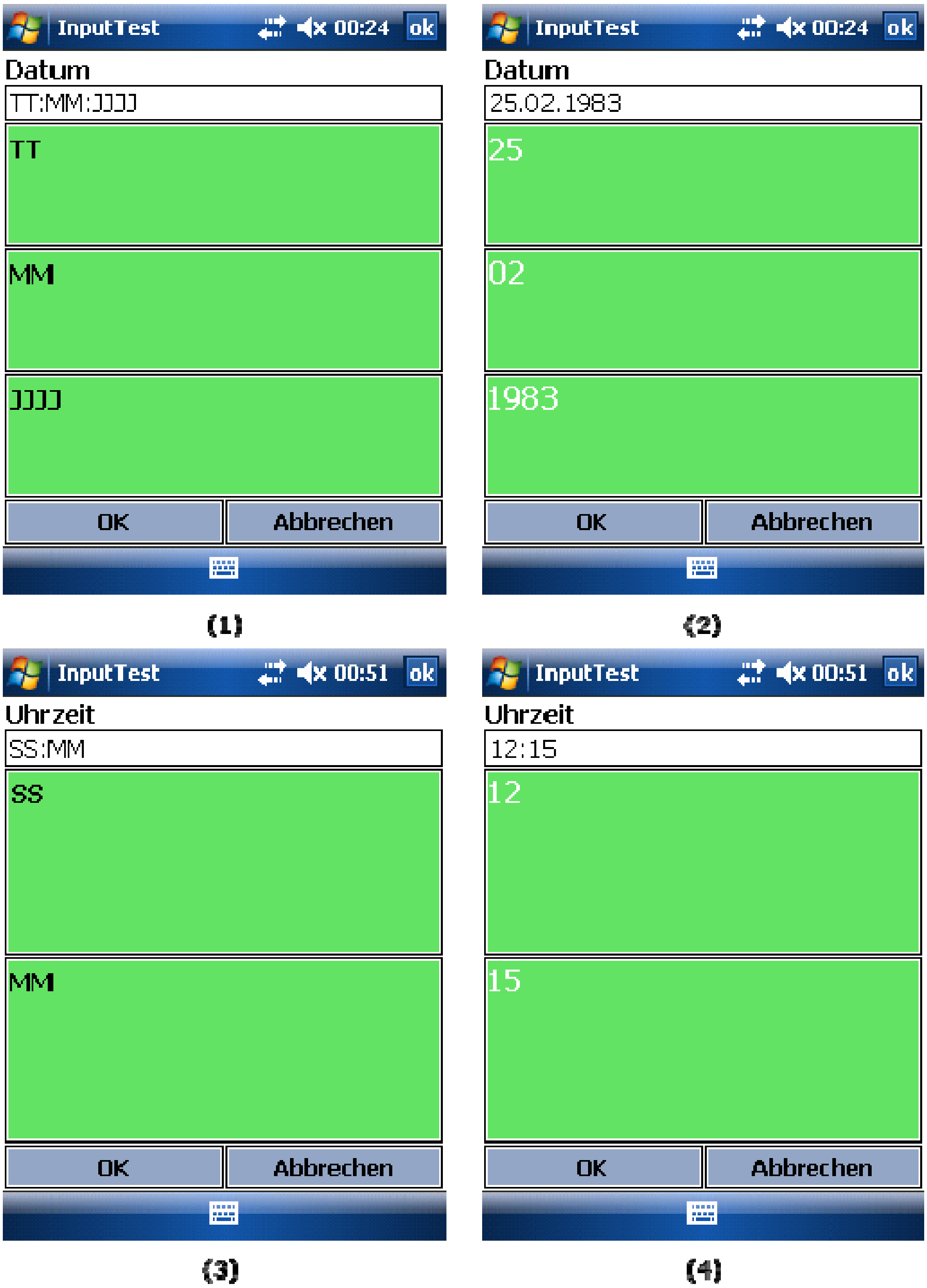


Figure 19: Very early drawn prototypes for guided handwriting input dialogs. (1) shows the dialog for entering a date, (2) for entering a time.

To get a maximum surface for handwriting input, each section was stretched to its maximum. Left-handed users will have the same surface as right-handed users for drawing their handwriting into the sections.

The headers (in Figure 19; Tag, Monat, Jahr, Minuten, Sekunden) got removed due to serve left-handed users in better way.
Instead of the headers, the placeholders for the data (TT, MM, JJJJ, and SS) are placed on the left side of each input section (Figure 20).
Moreover, recognized data is shown instead of the placeholders for each section. (Figure 20)



(1)
Datum
TT:MM:JJJJ
TT
MM
JJJJ

(2)
Datum
25.02.1983
25
02
1983

(3)
Uhrzeit
SS:MM
SS
MM

(4)
Uhrzeit
12:15
12
15

Figure 20: Prototypes for guided handwriting input dialog for date.
(1) shows the dialog for date input at the start. (2) shows the filled dialog for date input. (3) shows dialog time input at the start. (4) shows filled time input dialog.

3.7. Guided Date Handwriting Input Dialog with Century Button

We found that the input of the year is the most critical part, because the possibility of a recognition error on a word with the length of 4 is higher than on a word with length one or two.
Moreover, the input of day and month is more limited by a dictionary than the input of a year.

Therefore, we developed a special solution for inputting a date. Valid years are in the range of 1900 and 2099.

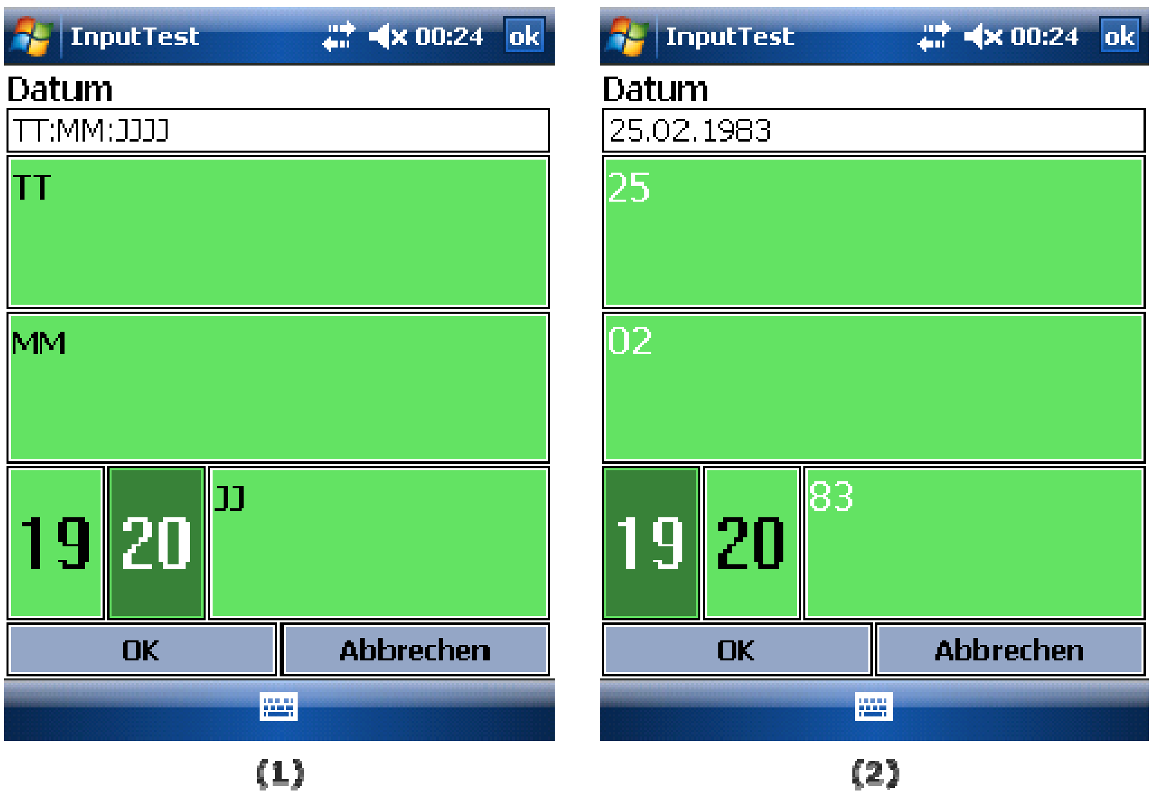


Figure 21: Guided handwriting dialog for imputing a date with century button.
(1) Dialog at the start, (2) Dialog filled with data

The century part (first two numbers) of a year is represented by two buttons (Century Buttons).
At the start, the "20"-Century-Button is activated, the year part of the result-edit-field is "20JJ" (Figure 21, (1)). This is, because we thought that years starting with "20" are more often inputted than years started by "19".

To change the century, the user only has to tip on the required century button.

Because of the constant colour scheme, the active century button is coloured in the darker version of green.

3.8. Character Recognition Dialogs

Due to problems described in Chapter 3.5.2 with recognizing whole words, there was the idea for character recognition. One character after another will be inputted.
Therefore, if a letter is recognized wrongly, the user has the chance to delete it immediately with one click on the backspace button, without any other interaction.
This is more comfortable and not as time consuming as pointing the cursor within a wrongly recognized word, deleting the wrongly recognized part and replacing it with the correct letter(s).

As described in Chapter 2.8.1, there is external and internal segmentation (Tappert et al., 1990). Segmentation could be a possible error source; therefore, it could lead the recognition to better accuracy if the segmentation is done externally, by the user, writing separate characters.

Written in Chapter 2.16, decreasing the number of alternatives lead to better recognition accuracy. Therefore, the set of valid letters has to be restricted to upper or lower case letters.
Upper case letters are bigger and consist of two strokes in average, while lower case letters are smaller and consist of one stroke in average (Chapter 2.7).
Therefore, only lower case letters are allowed to input by the user.

To input upper case letters, some interaction by the user via the interface has to be done. In the following chapters there are some approaches to do so.

3.8.1. Dialog

All input dialogs should fit the constant colour scheme and, most important, their appearance should be the same.

Therefore, we defined the position of some basic components (Figure 22)

- **Headline Section**, the name of the field which will be edited
- **Result Section**, the recognized letters will be displayed within an Edit Field
- **Command Section**, command buttons are displayed in this area (Show Keyboard, Space, Backspace, New Line, Left/Right Button)
- **Writing Section**, user will draw the handwriting in this area
- **Closing Section**, OK and Cancel button will close the dialog (with or without saving the changes)

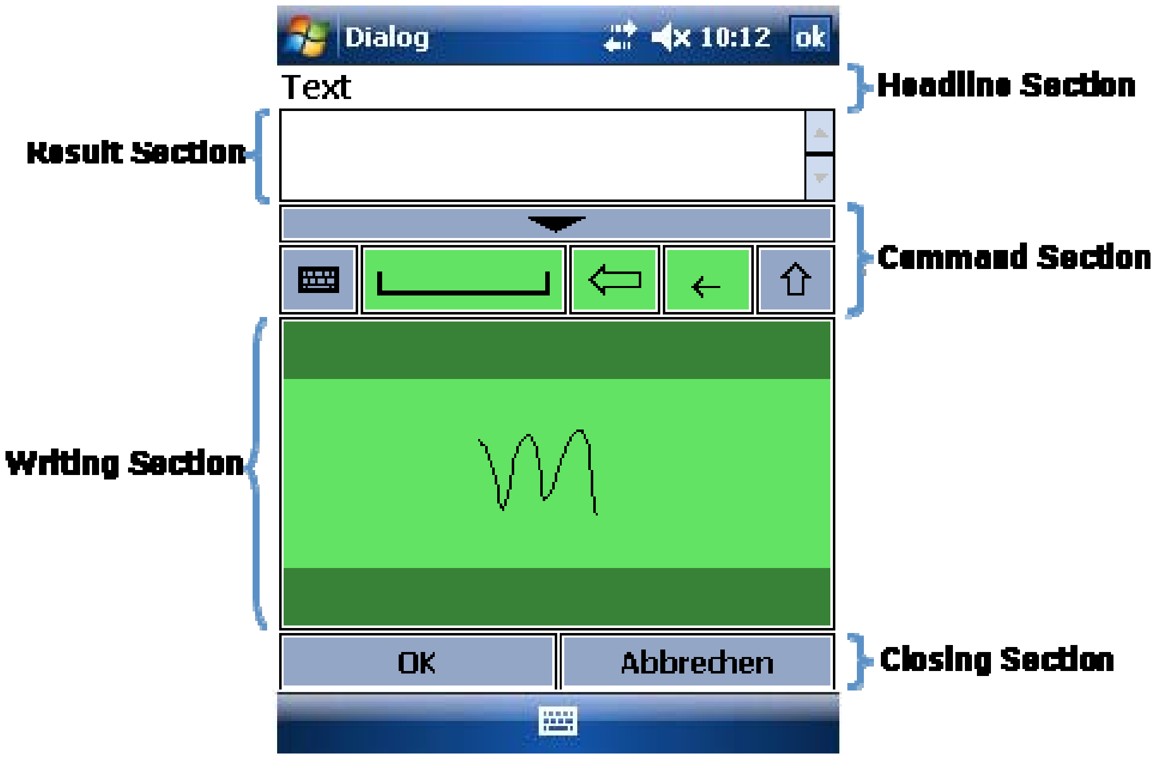


Figure 22: Basic component positioning

The position of the command section was chosen between Result Section and Writing Section, because during an evaluation (Cognitive Walkthrough) we found out, that users are checking the recognized letters during typing, and therefore, the view of the user is focused on the upper part of the screen.

The OK and Cancel buttons within the Closing Section are placed in the other way than the design guidelines would recommend (Confirm Button on the right side) (Zwick et al., 2005). Due to design guidelines of FERK Systems, the Confirm button is placed left, whereas the Cancel button is placed on the right hand side.

3.8.2. Text with Upper/Lower Case Button

This dialog was the first approach to give the user the possibility to choose upper or lower case letters to input and decide the case by a button.

This approach is derived from a Nokia 6120 Navigator's (Nokia) SMS input. The decision whether it is an upper or lower case letter is done by a button. There is also the possibility to type in text in printed letters (only upper case letters).

The flow and functionality of this button is shown in Figure 23.

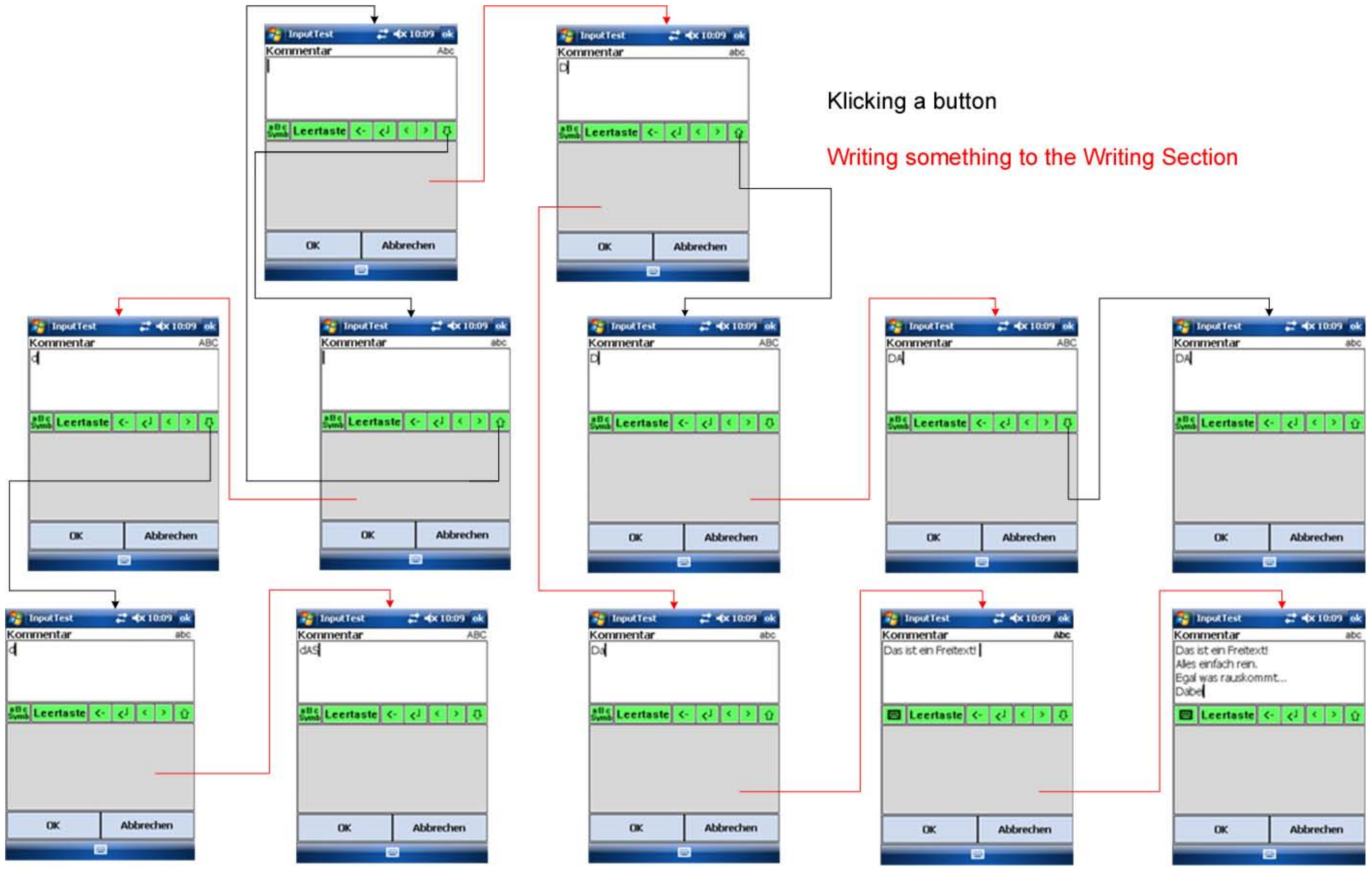


Figure 23 Flow Chart of the Upper-Lower Button Functionality

3.8.3. Text with Upper/Lower Case Area

In this dialog, the Writing Section is parted into two pieces by a horizontal line (Figure 24). Letters, with first stroke started within the upper part, will be displayed as upper case letters in the Result Section – starting within the lower part, they will be displayed as lower case letters.

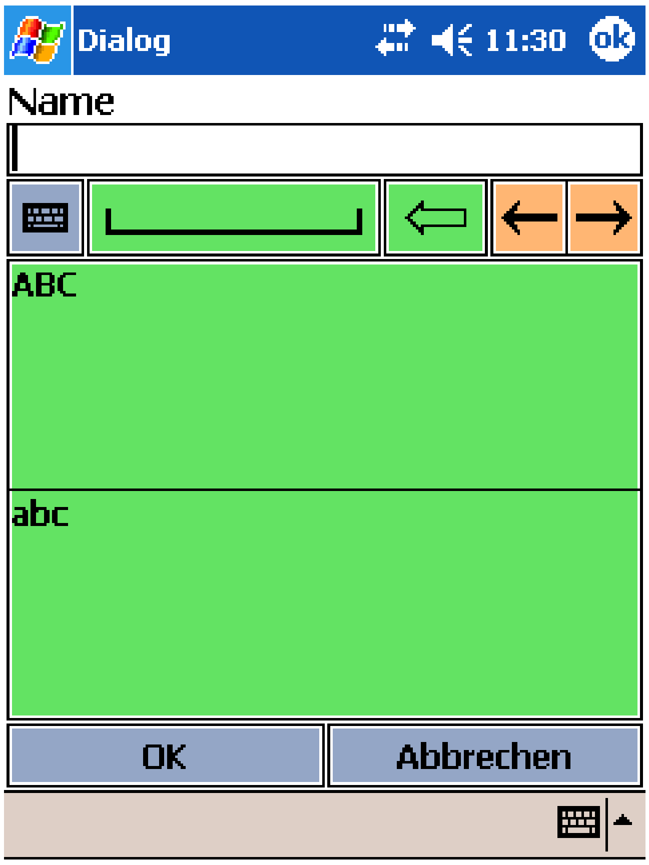


Figure 24: Text Input with Upper/Lower Case Area

In the best case, the user does not move his/her hand and the stylus to the upper or lower section and lets the hand remain in the middle area. Also inputting printed letters (only upper case letters) is possible.

3.8.4. Text with Upper/Lower Case Space Button

The basic idea of this dialog is that the user can write wherever he/she wants within the writing area.

If there is a need for distinction between upper and lower, it can be decided after writing a word.

There are two space buttons on the dialog (Figure 25).

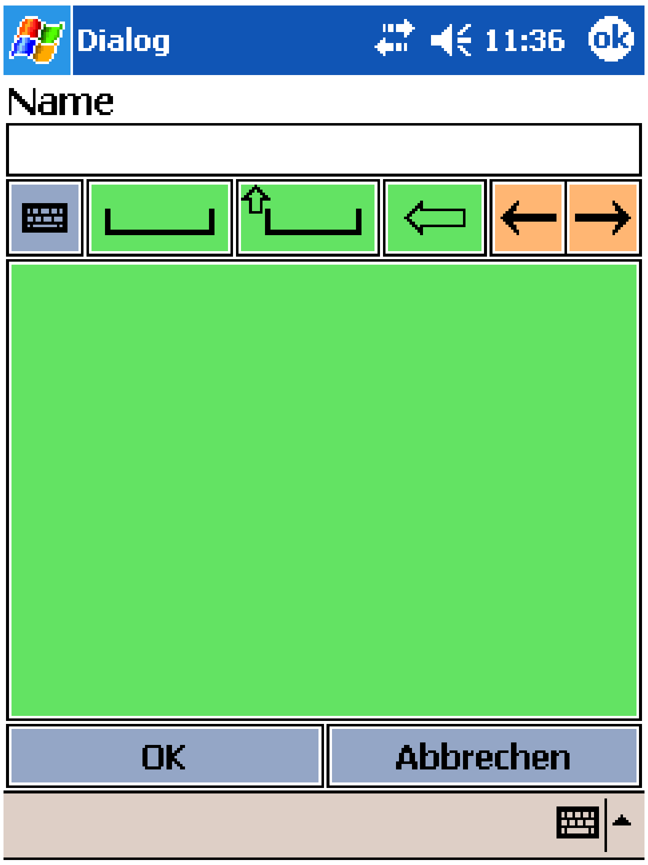


Figure 25: Text Input with Upper/Lower Space Button

If the user clicks on the space button with the upper case arrow, the first letter of the word just inputted will change to upper case and a space will be added.
If the user clicks on the normal space button, only a space will be added.

This dialog allows very fast writing, because the user does not have to change the position of the stylus, except when he wants to change the first letter of an inputted word.

This method gives the user the chance to decide if fast writing (without capital letters) or correct writing (with capital letters) is preferred or needed.

The negative side of this dialog is the fact, that no words in printed letters can be inputted (shortcuts, abbreviations) and the way to get a capitalized word might be confusing at the beginning.

3.8.5. Problem with upper case solution

One of the solutions in the previous two Chapters was thought to be the best solution for inputting upper case letters. Therefore, two tasks inputting text to the system, one with the upper space button solution and one with the upper/lower case area dialog were performed by six first responders. They were watched during the tasks.
The users had problems with the upper/lower case area because it was unnatural to them changing position for the beginning of a character.
The solution with the upper space button was very confusing to the users because of changing the first letter of a word after writing the word.
There were problems with both solutions, therefore, a new solution for the upper/lower case problem had to be found.

3.8.6. Text with Upper Case Button

This solution is very similar to the upper/lower case button solution in Chapter 3.8.2. The difference is that in this solution there is only an upper case button, it never switches to a lower case button.

Normally, the upper case button is displayed in Enabled Status (⇧). The recognized letters will be displayed as lower case letters in the input field.

Clicking on the upper case button changes the button to InUse Status (⇧). This gives the user the information that the letter recognized next will be displayed as upper case letter in the input field.
After recognizing one letter, the button switches back to Enabled Status and the letter recognized next will be displayed as lower case letter.
Clicking on the upper case button before recognizing the next letter will switch the button back to InUse Status. Now all the following letters recognized will be displayed as upper case letters, until clicking on the button again.

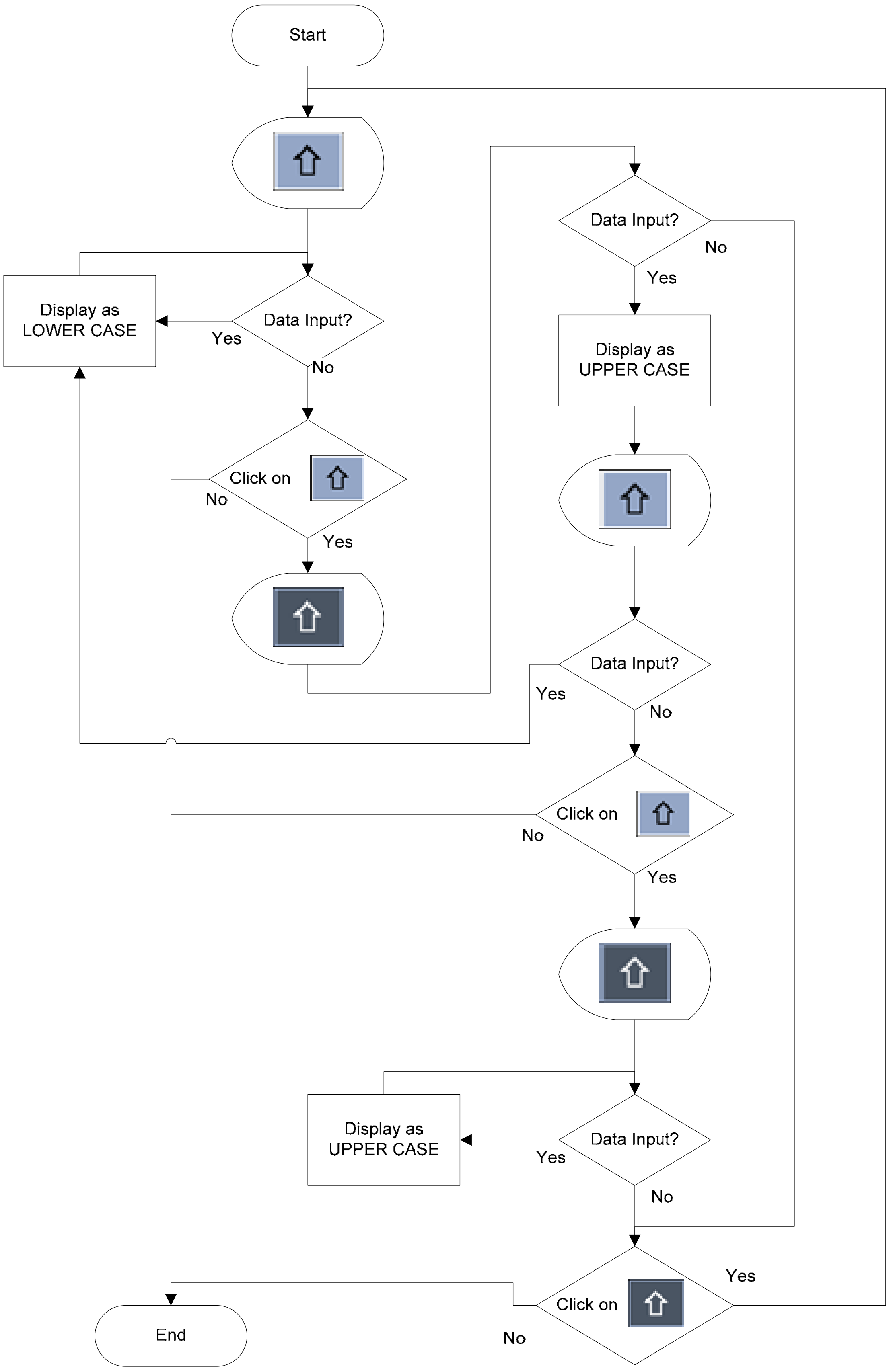


Figure 26: Flow chart of the functionality of the upper case button

3.8.7. Multiline Text

This dialog is based on the single line letter handwriting recognition dialog. The text field's height is increased to display three lines of text, and two additional buttons are added to the given dialog (Figure 25).

These new buttons are a new line button (adding a new line to the text), and the en/unroll button, to view more than 3 lines of the text box. (Figure 27)

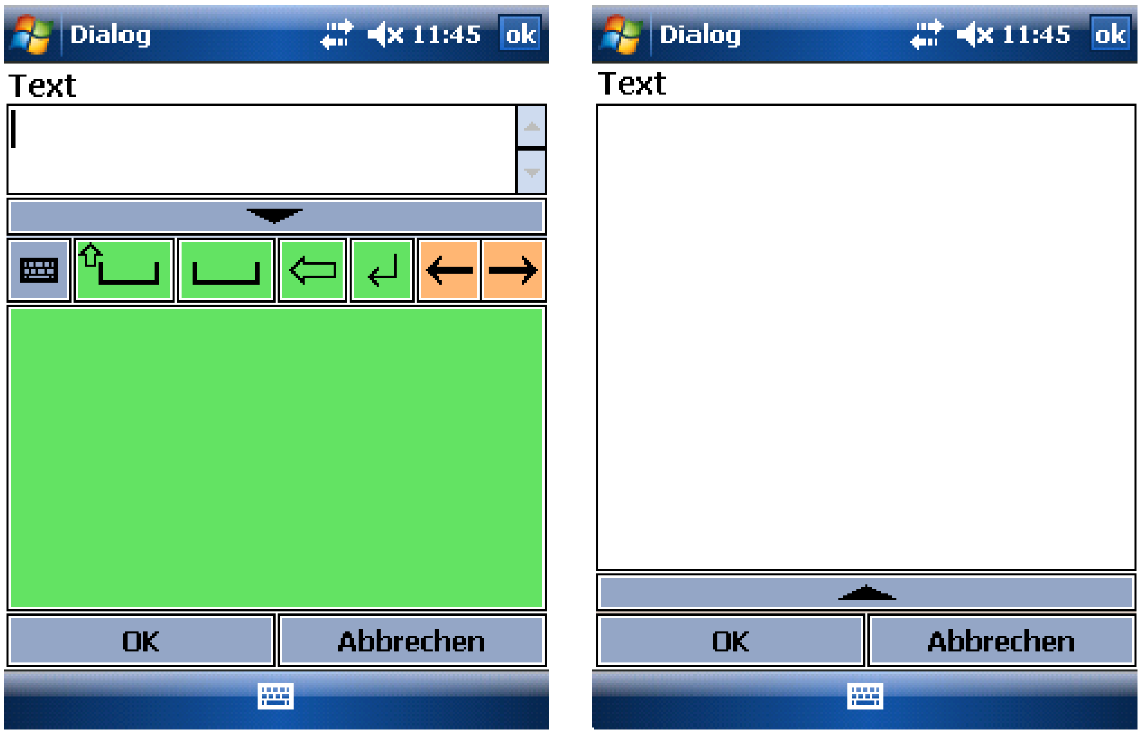


Figure 27: Multi Line Text Input

If the user wants to place the cursor somewhere in the multi line text (which is outside the view of the three line input field), he clicks on the unroll button, and the text size of the input field will increase and fill the screen. The Command Section (except the un/enroll button) and the Writing Section gets hided. The unroll button changes to the enroll button.

After that, the user positions the cursor with the stylus.

To get back to writing, the user clicks on the enroll button. The size of the input field goes back to three lines, the Command and Writing Section is displayed and the enroll button changes to unroll button.

The cursor remains on the changed position and is displayed in the input field.

3.8.8. Back Door

There is also a Back Door in the interface, if a user can not type in a letter with handwriting recognition. In this case, the user clicks on the keyboard button (). The virtual keyboard will be displayed () and the user can enter the letter. After entering the letter, the virtual keyboard is hidden (also by clicking on while the keyboard is displayed).

The Back Door is implemented for all dialog types (Text, Numbers, Time and Date).

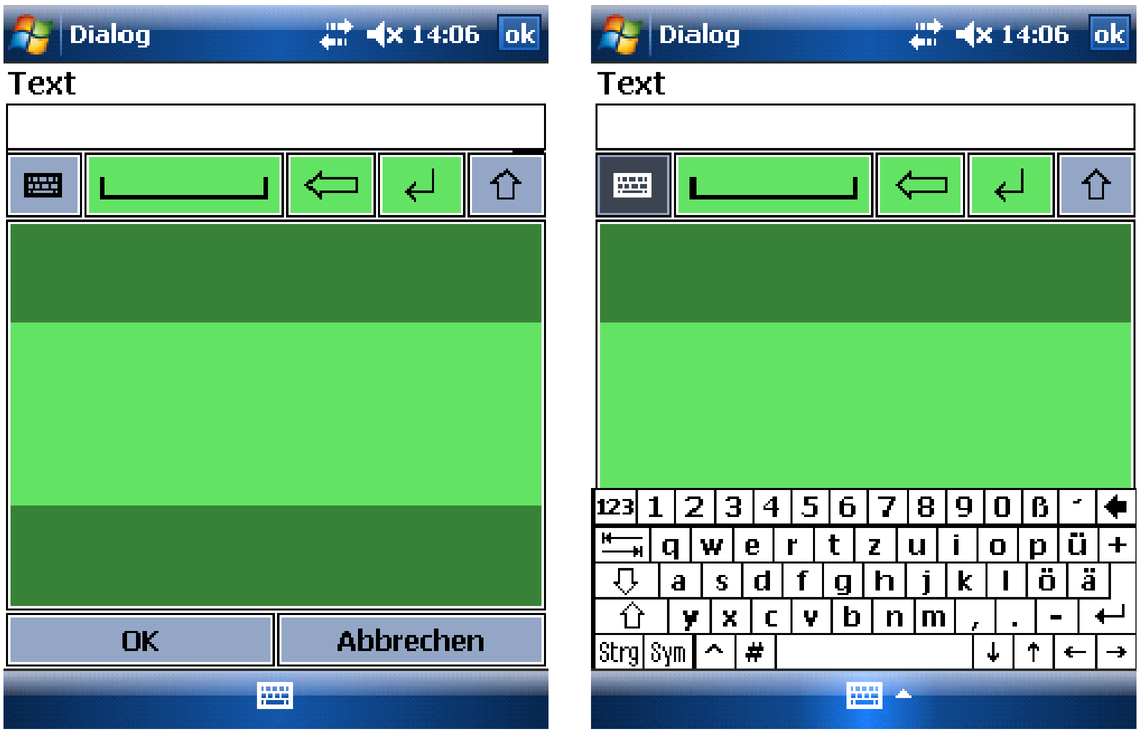


Figure 28: Back Door, the virtual keyboard

3.8.9. Final Text Component

The final text input component is a result of all the other prototypes and of the iterations which were done within the project.

The basic layout of the dialog is realized as it is described in Figure 22. Only the command section has changed.

The buttons for navigation actions to navigate to the left or to the right were removed, because of the decision that placing the cursor within the text has to be done by placing the cursor with the stylus.
Therefore, the buttons for input actions and command actions are bigger and easier to hit.
The Writing Section also changed. Phatware defines the best recognition results for letters which have a height of 80px (Figure 9), which is presented as the brighter area (Figure 29, Figure 30).

The user is called to pay attention to the Result Section (input field) to check if the written letter is recognized correctly. There is no need for the eyes of the user to be focused on the writing, because the user knows how to write letters anyway. Moreover, because of the Adaptive Timeout (Chapter 3.9), the user is able to write fluently and does not have to wait until the recognition starts.

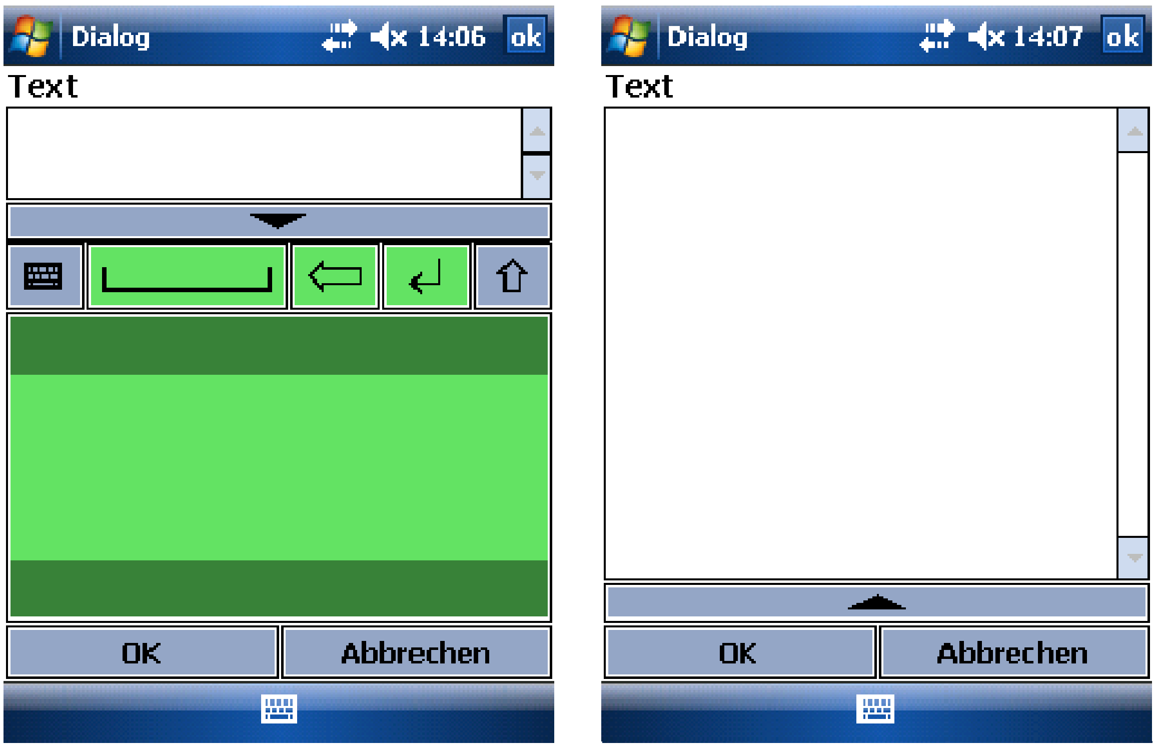


Figure 29: Final Multi Line Text Dialog

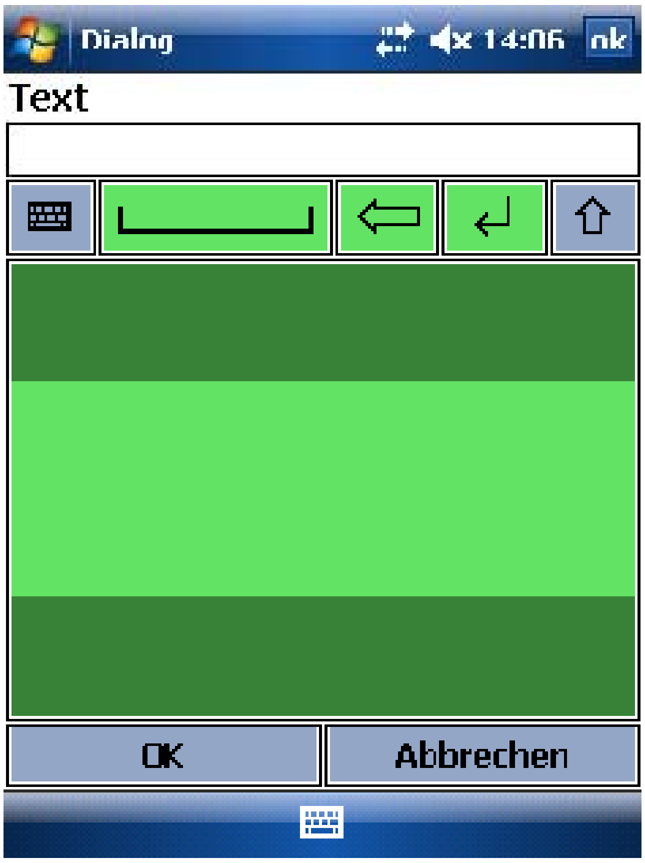


Figure 30: Final Single Line Text Dialog

3.8.10. Numbers, Date, Time

Due to a constant design of the dialogs, these dialogs are based on the Final Text Dialogs. Buttons which are not needed are not deleted from the dialog, they are still there, but disabled.
Every dialog has the same appearance, but for these dialogs, the button space, new line and upper case are disabled.

For entering numbers, date and time, the Calligrapher is configured with special dictionaries, containing only numbers and for entering the numbers “+”, “-“ and “,”.

Wrong dates or times (e.g. 25:74 or 31.02.1993) cannot be inputted to the system - the system avoids this.

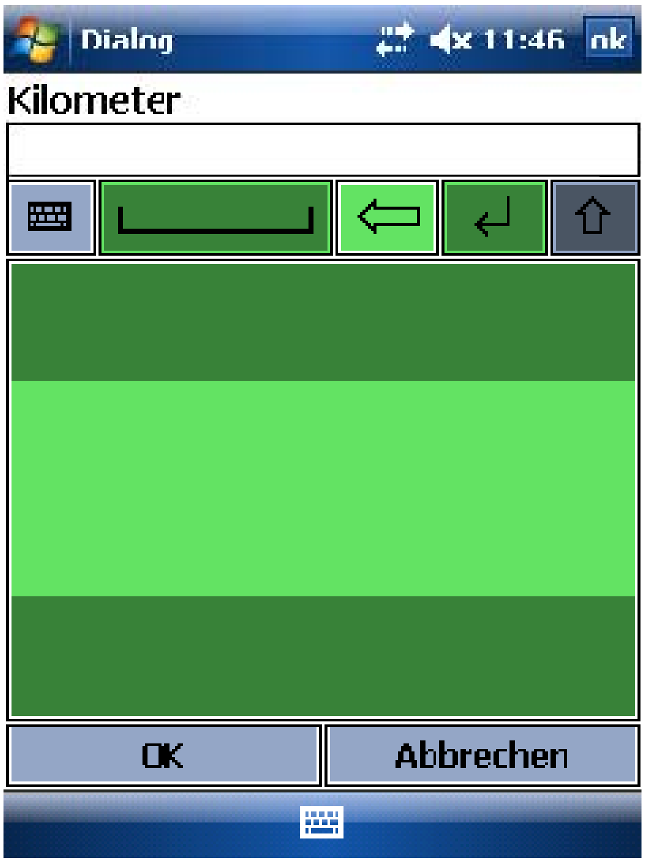


Figure 31: The final dialog for entering numbers

3.8.10.1.Date and Time Dialog

At these dialogs, the input field is filled with a mask, TT.MM.JJJJ for date and SS:MM for time.

Each inputted number replaces the placeholder next to the cursor.

Table 10: Example of entering a date

Input field	Inputted number or interaction
|TT.MM.JJJJ	2
2|T.MM.JJJJ	5
25.|MM.JJJJ	0
25.0|M.JJJJ	3
25.03.|JJJJ	⇦
25.0|M.JJJJ	2
25.02. |JJJJ	2
25.02.2|JJJ	0
25.02.20|JJ	1
25.02.201|J	0
25.02.2010|	

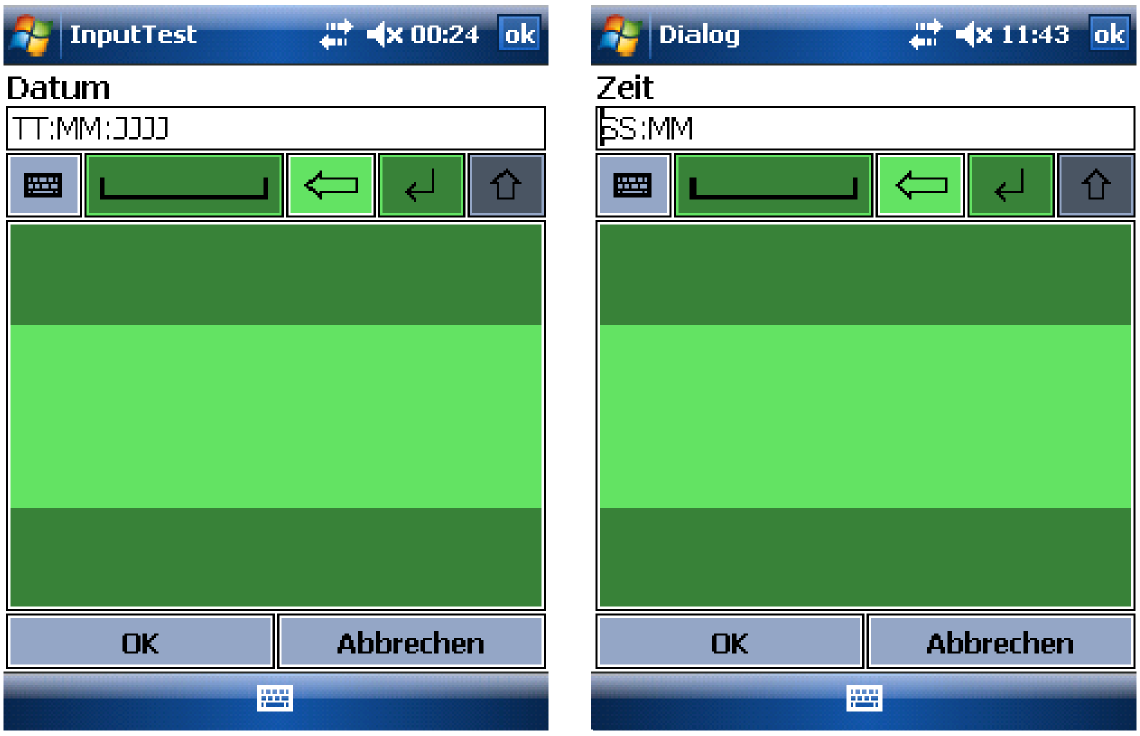


Figure 32: Final date and time dialog

3.9. Adaptive Timeout

Letters could consist of more than just one stroke (e.g. a handwritten “l” consists of one stroke, while an “F” consists of three strokes), therefore it is very important to increase

the speed and satisfaction of the user is to optimize the time between the ending of the last stroke of a letter and the start of the recognition.
But when has the user finished inputting strokes for one letter? Groner describes a symbol completed if a specified time elapses after the end of the most recent stroke (Groner, 1966).
The elapsed time after the last stroke of a letter we call timeout.

It is impossible to define a general timeout, because of users' different speed of writing letters, e.g. a user who writes fast will be slowed down by the system, if the timeout is set too high, whereas a user who writes slowly could not be able to input letters which consist of more than one stroke because of a timeout too short.
Therefore, we developed a variable timeout, which automatically fits the user.

The most important data for calculating the timeout is the passed time between two strokes (TBS, Time Between Strokes), if a letter consists of more than one stroke.
The letter "F", for example, consists of three strokes, therefore two times between the strokes can be measured (Figure 34), while the letter "S" consists of only one stroke (Figure 33).

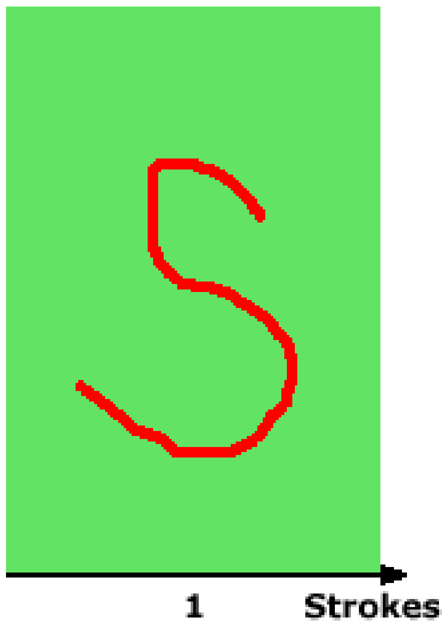


Figure 33 Letter "S"consists of one stroke

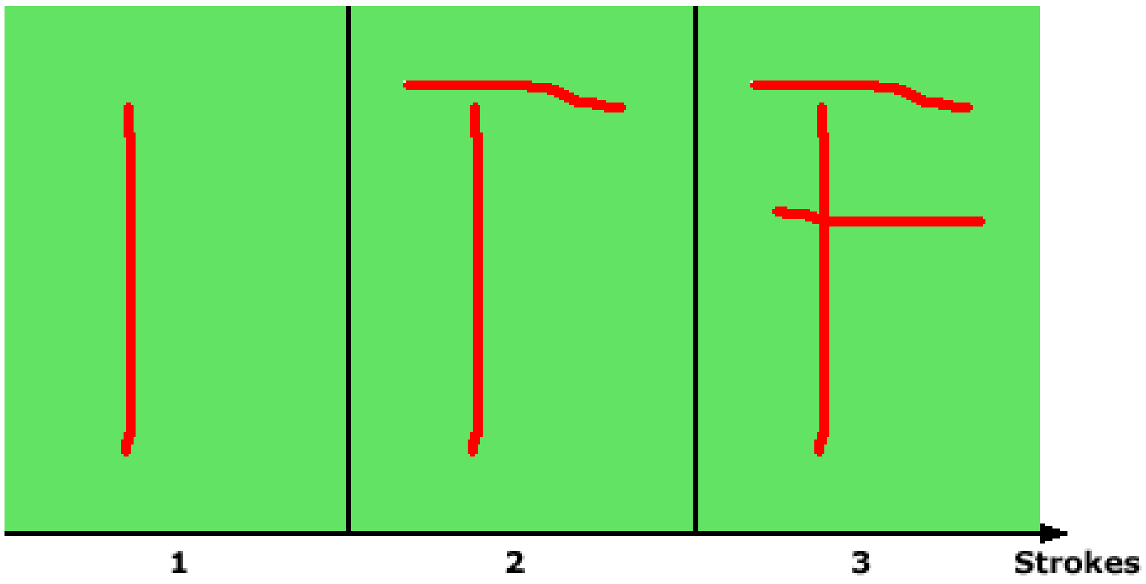


Figure 34 Letter “F” consists of three strokes

The following formula calculates the optimized timeout:

Equation 5: Optimized timeout calculation if there is an actual TBS

$$T = \frac{\sum_{i=1}^{n} s(i)}{n} * \frac{X}{100}$$

T	is the resulting timeout;
s_1	is the actual average TBS (for a new user, it is an initial value);
$s_2 \ldots s_n$	are the last n measured TBS;
X	is a factor;
n	numbers of measured TBS, maximum 10

Equation 6: Optimized timeout calculation without actual TBS

$$T = \frac{\sum_{i=1}^{n} s(i)}{n} * \frac{X}{100}$$

T	is the resulting timeout;
$s_1 \ldots s_n$	are the last 10 measured TBS;
X	is a factor;
n	numbers of measured TBS, maximum 10

The system automatically stores the last 10 measured TBS. Thus, , the system adapts the TBS to the new speed of writing if the user does not use the common stylus or biro for writing and the speed of writing changes (greater or smaller TBS).

3.10. Correction Intervention

Calligrapher Handwriting Recognition does not adapt recognition to users handwriting because of static Fuzzy-Neuronal Nets (Strenge, 2005).
This is a possible problem for users with weird handwriting. We found out about this problem by watching a user trying to type in an “a”. Figure 36 shows the way the user tried to write his “a”. These letters are called Wrong-Recognized-Letters.

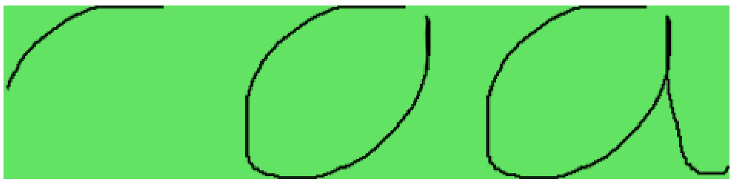


Figure 35: Handwritten “a”, Calligrapher recognizes an “a”

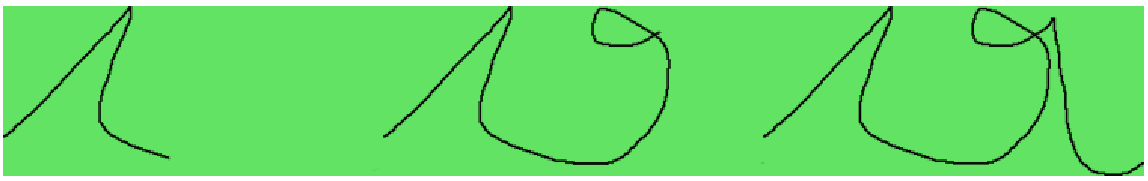


Figure 36: Wrongly-Recognized-Letter; Handwritten “a”, Calligrapher recognizes “ir”

In this case, there is no direct solution because of the static Neuronal Nets no adaptation of the handwriting recognition to the user’s writing is possible.

Basically, the handwriting recognition engine returns a list of possible results with its possibilities (weights). Henceforth, such a result list is called a schema.
For example:

u	52
n	47
k	37
M	35
A	22

This means that the inputted letter is an “u” with a possibility of 52%, a “n” with a possibility of 47% …

Each user has to accomplish a calibration process when using the system for the first time. (3.11, Calibration). In this process, the system knows which letter has to be inputted by the user. If the recognition does not return the expected letter as first choice, we know that the user’s handwritten letter was recognized wrongly. Because of the knowledge of which letter was expected, the recognized schema can be stored.

E.g. Expected Result: a

u 52
n 47
k 37
M 35
A 22

This example means, that this is a Wrong-Recognized-Letter, its schema and expected result will be stored as an “a”.

After the calibration process, there are schemas stored, if there were letters causing problems (like Figure 36).

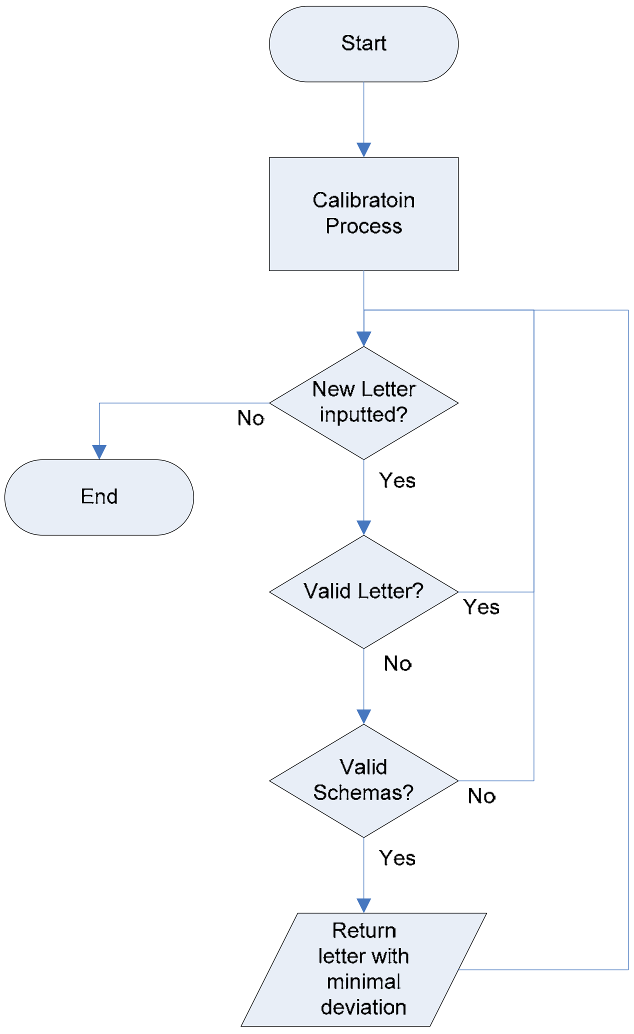


Figure 37: Correction Interventions as flow chart

During writing, the schemas (received from handwriting recognition engine) will be compared to the data which are collected in the calibration process.

“Valid Letter?” in Figure 37 compares the inputted letter’s highest weight with the average weight of the letter for this user. If the weight of the inputted letters is not less than the average weight of this letter for this user, there is a definite correctly recognized letter found.
If the weight of the inputted letter’s highest weight is less than the average of the letter for this user, the schema will be compared to the stored schemas.

Comparing the inputted schema to the stored schemas (“Valid Schemas?”, Figure 37) is divided into

- Checking validity of a schema
- Calculating the average deviation.

This is done for each stored schema.

The validity of the inputted schema to a stored schema is checked the following way:
If the inputted schema consists of

- **one letter**, it is **NOT VALID**
- two letters, **two** letters have to be within the compared stored schema, to become **VALID**, which means **2/2** (2 conformities out of 2 possibilities) are required
- three letters, **2/3** are required
- four letters, **3/4** are required
- five letters, **3/5** are required

After checking the validity of the inputted schema, there are 0…n valid schemas. To find out the least deviant stored schema, the average deviation for each of them to the inputted schema is calculated.
To calculate the average deviation, the difference of the weights of the same letter in the inputted schema and compared stored schema will be summed up and will be divided by the count of same letters.
For example:

inputted schema			**Comparison schema**		**Deviation**
u	52		k	41	4
n	47		M	35	0
k	37		n	31	16
M	35		h	26	
A	22		m	22	
Validity: 3/5 → VALID					
Average Deviation:					(16+0+4)/3 = **6**

Figure 38 comparing two schemas and calculating an average deviation

The whole process returns the schema's resulting letter of the schema with the minimal average deviation.
The returned resulting letter will be set on top (in front of the others) of the list which was returned by the handwriting recognition.

3.11. Calibration

The key issue of the calibration is to find out data of the user about each letter which can be inputted.
After the calibration, the system knows the following data of the user:

- Average weight for each letter which definitely identifies a letter
- Schemas of Wrong-Recognized-Letters (see Chapter 3.10)
- Optimized Timeout

To get there, the user has to input each letter at least 2 times.

The first problem is that the system needs to know which letter the user has to input. Therefore, the system orders the user to input the letters, one after another (Figure 39).

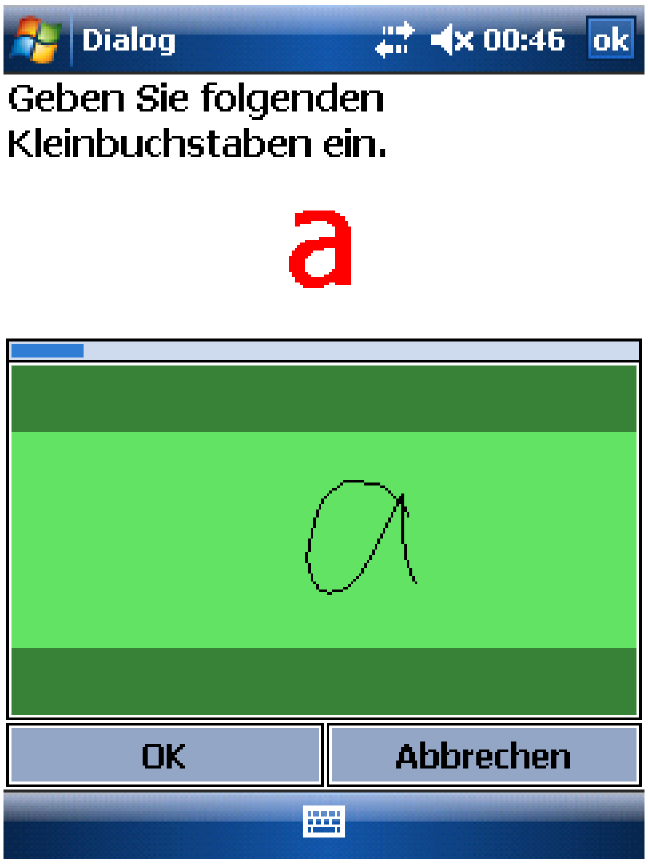


Figure 39: Screen during calibration

The set of letters which are calibrated consists of all German lowercase letters:

- “ä”, “ö”, “ü”
- “a”…”z”
- “ß”

The system orders the user to input the letters exactly in this order. The letters “ä”, “ö”, “ü” are chosen as the first letters because they consists at least of 2 strokes each letter. Therefore, after these three letters there is already an adapted average timeout for the user calculated. The 30 letters are added to LIST, 10 times (Figure 40 (1)).
Each letter has to be inputted by the user at least two times and usually 10 times (except the user inputs a wrong letter).

Before we decided to use the solution that the letters are displayed in alphabetical order there was another thought to avoid wrong inputs by the user.
Many people are used to the alphabet, therefore the user might be writing without looking on the screen to see which letter is requested.
The other solution was to display the letters in a random way.
But this solution was not used, because of the fact that if the user inputs a text to the system, the letter which comes next is also known to the user.
Therefore, it is no problem that the user knows that after an “a” comes a “b” in the alphabet.

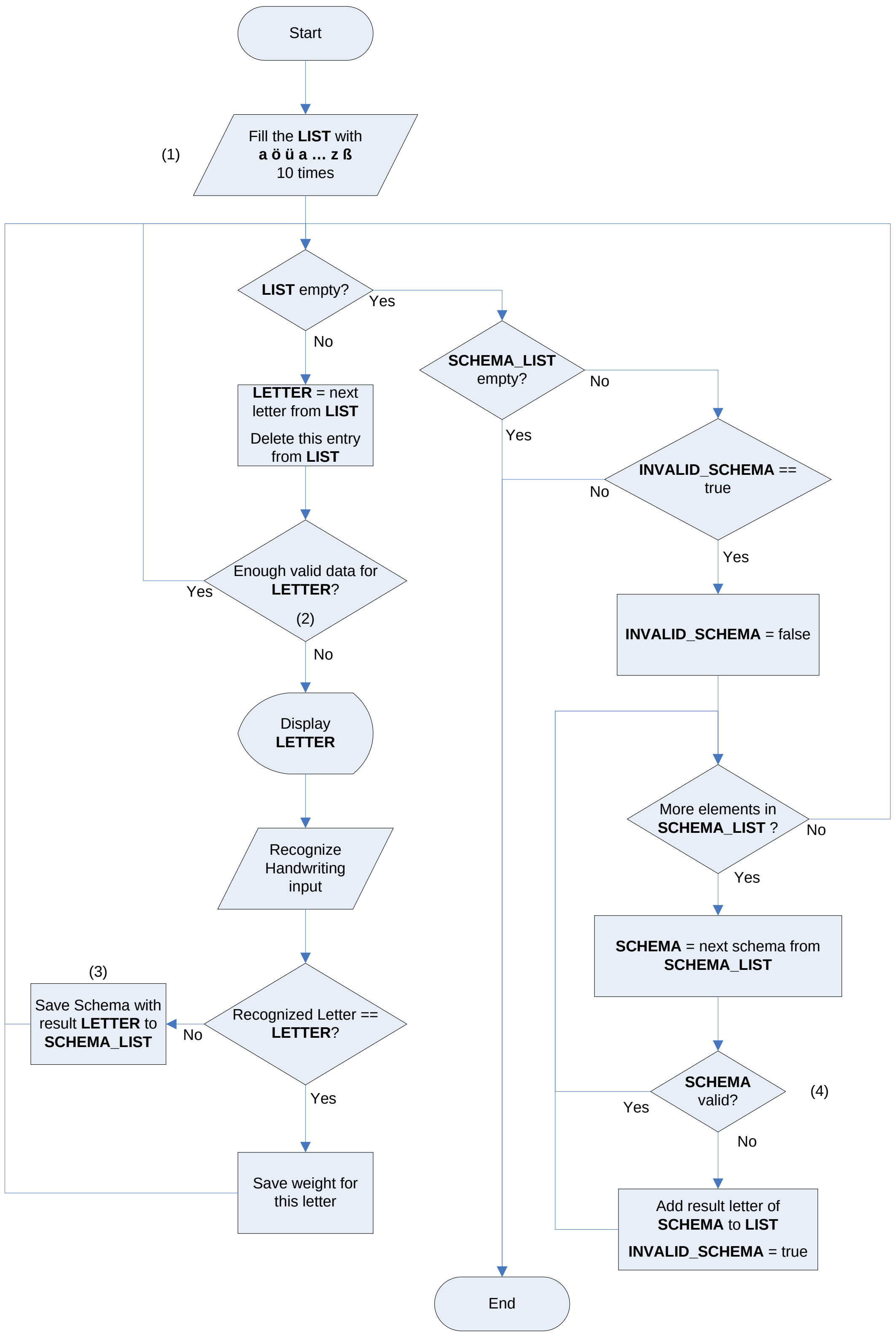

Start
(1)
Fill the LIST with a ö ü a … z ß 10 times
LIST empty?
Yes
No
SCHEMA_LIST empty?
No
Yes
LETTER = next letter from LIST
Delete this entry from LIST
INVALID_SCHEMA == true
No
Yes
Enough valid data for LETTER?
(2)
Yes
No
INVALID_SCHEMA = false
Display LETTER
More elements in SCHEMA_LIST ?
No
Yes
Recognize Handwriting input
SCHEMA = next schema from SCHEMA_LIST
(3)
Save Schema with result LETTER to SCHEMA_LIST
No
Recognized Letter == LETTER?
Yes
SCHEMA valid?
(4)
Yes
No
Save weight for this letter
Add result letter of SCHEMA to LIST
INVALID_SCHEMA = true
End

Figure 40: Calibration as flow chart

The decision “Enough valid data for LETTER?” (Figure 40 (2)) find out if enough valid data for the given letter is already inputted:
Valid letter means that the letter which was requested by the system to input by the user has the highest weight in the resulting list of the handwriting recognition.
Valid enough in the context of Figure 40 (2) means:

- **2/2**: **2** valid recognitions out of **2** tries
- **2/3**: **2** valid recognitions out of **3** tries
- **3/4**: **3** valid recognitions out of **4** tries
- **3/5**: **3** valid recognitions out of **5** tries
- **4/6**: **4** valid recognitions out of **6** tries
- **5/7**: **5** valid recognitions out of **7** tries
- **6/8**: **6** valid recognitions out of **8** tries
- **6/9**: **6** valid recognitions out of **9** tries

If the highest weighted letter of the resulting list of the recognition engine is not equal to the letter requested by the system, the inputted schema with the expected result will be stored to SCHEMA_LIST (Figure 40 (3)).

After each entry in LIST (Figure 40, LIST Empty?) was processed, data for the user about

- For each letter the weights with which a letter is recognized (therefore the average weight)
- List of schemas which represent Wrong-Recognized-Letters

is stored.

But what if the user writes a wrong letter, e.g. the system requests an “s”, but the user writes a “c”? – Therefore, all saved schemas are checked:
If the highest weighted letter of a stored schema compared to the average weight of this letter for this user is valid, the schema will be deleted, and the resulting letter of this schema will be added to LIST once again. (Figure 40, SCHEMA valid?)
After checking all stored schemas, all elements of LIST will be ordered to input once again (if there are not enough valid inputs of this letter for the user).

For example:

Saved Schema, resulting letter "c"		**Valid data for "s"**
s	98	90, 94
5	30	Average 92

In this example, "c" will be set on LIST once again, because "s" with its weight 98 is greater than the average weight for "s" 92.

3.12. Continuous Calibration

Similar to the adaptive calibration (Chapter 3.11) a continuous calibration during normal text input is done.

Because of free text entry at this point, the system does not know if inputted letters are right or wrong.
We solved the problem the following way:
The system remembers interactions and its concerned letter while the user is inputting data (writing) in a queue of the size 10. Interactions are

- Deleting of a letter
- Writing a letter

If an inputted letter will not get deleted within the next 10 inputted letters, it is assumed that the letter was recognized correctly.
Therefore, every time, a new letter gets inputted, the interaction with its concerned letter is added to the queue.
If an interaction and its letter fall out of the queue, it will be checked if this letter was deleted within the last 10 interactions.
If the letter was not deleted, the weight of the inputted letter will be added to the weights of this letter for the user.
If the letter was not deleted, and the letter was a Wrong-Recognized-Letter, its schema will be added to the list of schemas for the user.

On closing the dialog, the steps above will be performed for remaining inputted letters, because on closing the dialog, the letters should be correct inputted letters.

3.13. Other Interventions on recognition results

3.13.1. Continuous adaptation of the set of valid results

As written in Chapter 2.16, keeping the set of alternatives which are valid results for the recognition as small as possible, will increase the recognition accuracy. Therefore, during writing there are three possible input states where results of the following categories are valid

- **State All:** numbers , letters and symbols
- **State Text:** letters and symbols
- **State Number:** numbers, “+”, “-“, “.”, “,”

At the beginning of a word or number, the input state is set to State All.
If the first recognized result is a number, the input state switches to State Number, and until the next space or punctuation mark, only numbers and some symbols are valid results.
If the first recognized result is a letter, it switches to State Text.
If the first result is a symbol, the input state will remain on State All.

So it is not possible to write a number within a word or a letter within a number.

3.13.2. Confusable Pairs

To deal with highly confusable pairs Chapter 2.16, we implemented some functionalities. Confusable pairs are identified as

- "r" "v"
- "t" "f"
- "2" "z"
- "5" "s"
- "O" "0" "o"
- "9" "g" "q"

Every time, the user deletes a character with Backspace (⇦), the system saves this character until the next interaction by the user. If the next interaction is an input, the just deleted character is ignored, if it is the highest ranked result in the recognition list.

This functionality is based on the assumption that the user acts on wrongly recognized characters as follows:

1. Points the cursor after the wrongly recognized character (if it is not already there)
2. Press Backspace button
3. Inputs the character once again

3.13.3. "O" and "0" (zero)

This is a special case which occurs when the user wants to input a capitalized "o" or a "0" (zero) as first character of a word or number.
The "O" and the "0" look very similar on the screen. Because it is the first character inputted, the input state (see Chapter 3.13.1) would be **State All**. Therefore, "o" and "0" is a valid result.
The functionality to solve the "O" – "0" problem is displayed in the flowchart in Figure 41.

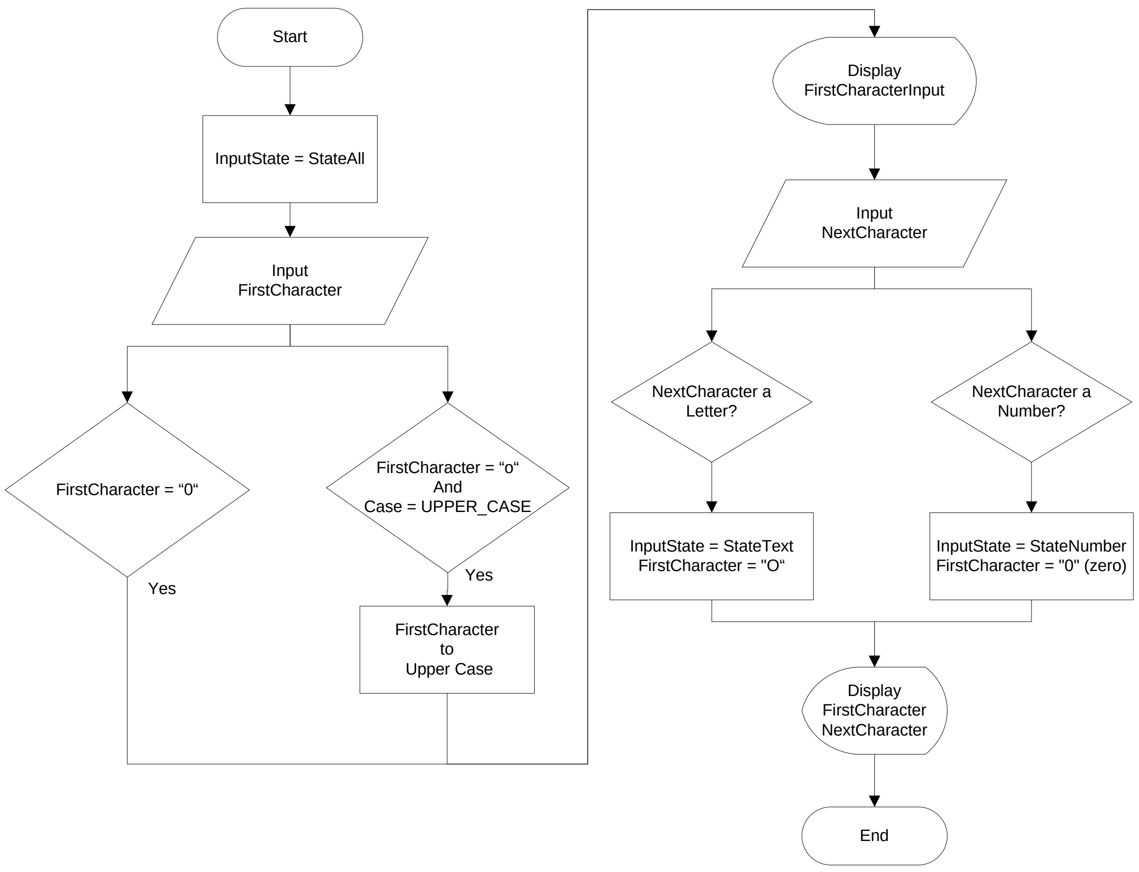

Start
InputState = StateAll
Input FirstCharacter
FirstCharacter = “0“
Yes
FirstCharacter = “o“ And Case = UPPER_CASE
Yes
FirstCharacter to Upper Case
Display FirstCharacterInput
Input NextCharacter
NextCharacter a Letter?
NextCharacter a Number?
InputState = StateText FirstCharacter = "O“
InputState = StateNumber FirstCharacter = "0" (zero)
Display FirstCharacter NextCharacter
End

Figure 41: Flowchart of the functionality to solve the “O”-“0” problem

4. Experiment

4.1. Prototype Application

4.1.1. Hardware

The device used for the prototype was an Asus MyPad A626 PDA. This device is equipped with an anti-glare touch screen display.
For typing on the touch screen, a stylus in the shape of a biro is used. Table 11 contains the technical specifications of this device.

Table 11: Specification of the PDA used for the experiment

CPU	Marvell XScale, 312MHz
Operating System	MS Windows® Mobile™ 6
Memory	256MB Flash ROM and 64 MB SDRAM
Display	3.5" Brilliant TFT LCD 65k full-colours, anti-glare 16-bit display QVGA, 240x320 px touch screen
Weight	158g
Physical dimensions	117 mm x 70.8 mm x 15.7cm

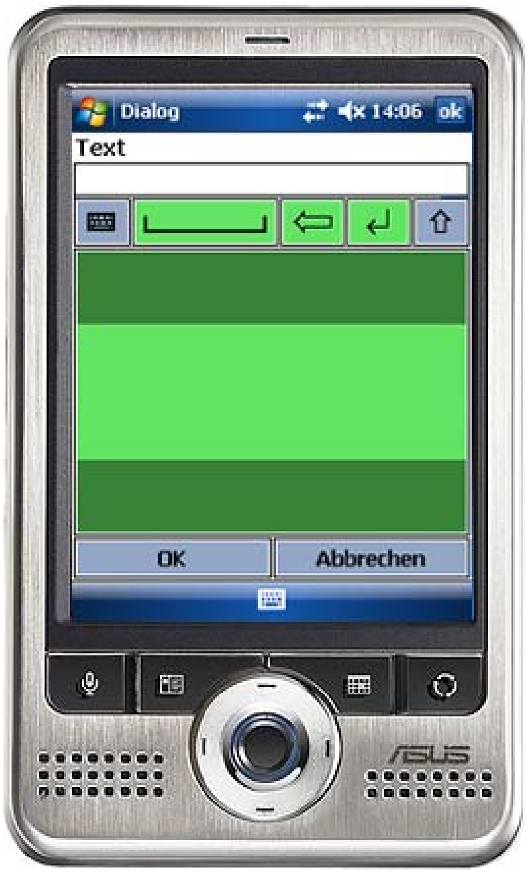


Figure 42: Used PDA with the text input Dialog

4.1.2. Software

The developed prototype for text input dialog is algorithm for optimized timeout and correction interventions described in Chapter 3.8.9, Chapter 3.9, Chapter 3.10 and Chapter 3.11 were the basic component of the experiment software.
There is also a text input dialog with a virtual keyboard implemented.

The application stores statistical data of the whole experiment.

- Each interaction by the user, e.g. inputting a character or using a button. For each of these, a timestamp is also stored.
- Time Between Strokes, Optimized Timeout, to view the adaptation to the user's speed
- Correction Interventions and Correction of confusable pairs, the originally recognized character and the finally displayed character is stored

The FACTOR for Equation 5, Equation 6, calculating the optimized timeout is set to 200 within the experiment. At the beginning, there are no TBS stored in the TBS queue; therefore 300 will be used as the initial TBS.
This means that the first timeout for each user is calculated with Equation 6 resulting in a timeout of 600 milliseconds.

The flow of the screens:

- Welcome Screen
- Description of the keyboard input followed by the dialog with virtual keyboard input (dialog to get familiar to the dialog, free text input)
- Description of the experiment for keyboard input followed by the dialog
- Description of the calibration and the calibration
- Description of the handwriting input dialog followed by the handwriting dialog input (dialog to get familiar to the dialog, free text input)
- Description of the experiment for handwriting input followed by the dialog
- Good-Bye Screen

The experiment application is designed in a way that the user can cancel the experiment at any time.

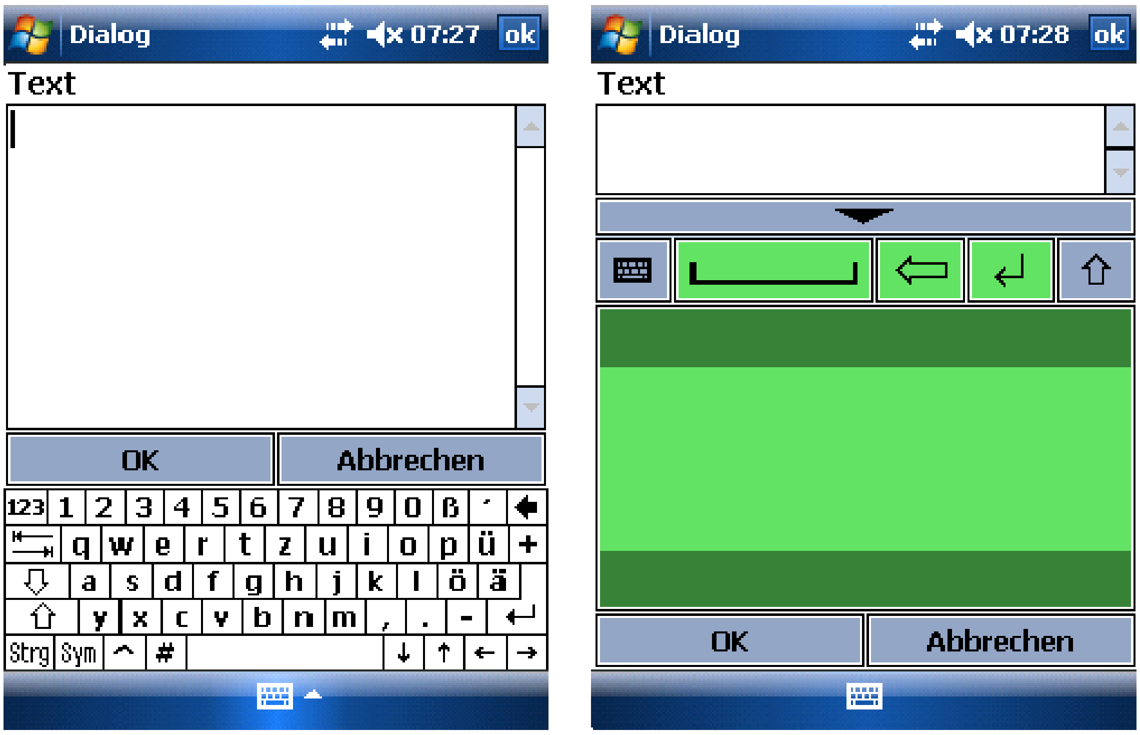


Figure 43: Screenshot of the virtual keyboard and handwriting dialog

4.2. Participants

Basically, participants in the experiment should work within health care. E.g. doctors, nurses, students of medicine and paramedics. In Austria, the bigger part of emergency medical services is done by the Red Cross. Paramedics of the Red Cross are professionals and volunteers.

Because there are volunteers in health care too, everyone could be a potential participant.

Because of the fact that nearly everyone is familiar to a QWERTY keyboard layout two elderly people were also asked to perform the tasks of the experiment.

The users were also asked to fill out a background questionnaire (Table 12).

Table 12: Results of the Background Questionnaire

	Min	**Max**	**Average**
Age	20	85	33.76
Uses PC for … Years	0	25	12.38
Uses PC … hours a week	0	60	30.62

Gender	
Male	11
Female	10
Optical Aid	
Glasses	10
Contact Lenses	2
None	9
Handedness	
Right Hander	20
Left Hander	1
Experiences with Mobile Devices (Touch screen)	
iPhone	4
Samsung Mobilephone with Touchscreen	2
HTC Diamond 2	1
LG Mobilephone with Touchscreen	1
Blackberry	1
PDA	1
Other Mobile Phone with Touchscreen	1
None	10

11 Participants use a PC ≤ 30 hours a week, while 10 participants use a PC more than 30 hours a week.

4.3. Experimental Settings

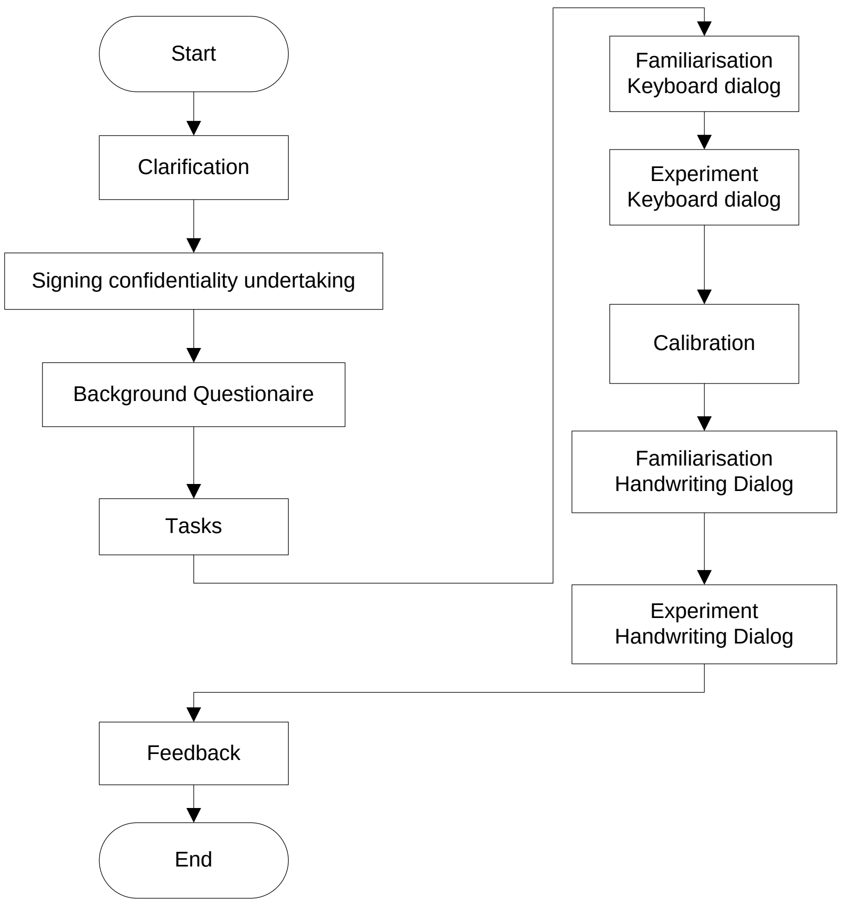

Start
Clarification
Signing confidentiality undertaking
Background Questionaire
Tasks
Familiarisation Keyboard dialog
Experiment Keyboard dialog
Calibration
Familiarisation Handwriting Dialog
Experiment Handwriting Dialog
Feedback
End

Figure 44: Flow Chart of the Experiment

The real life environment would be a seat in an ambulance car, documenting the rescue operation. To avoid negative effects on the work done by paramedics, the experiment is done at the base of the ambulance service in a quiet room.
The real life environment of a car is simulated there by sitting on a chair without armrests.
Simulating the environment of the experiment gives almost the same result as the experiment in real life (Kjeldskov et al., 2004)

Basically, the experiment application consists of two main parts, virtual keyboard and handwriting recognition.

Within these parts, there is a phase where the user can input words on his own, to get used to the system. This phase is not limited in time. The user can interact with the dialog as long as is needed in order to be familiar with it.

To measure accuracy, the experiment's tasks have to be text copy tasks (MacKenzie and Soukoreff, 2002), where the text which has to be inputted by the user is given. The user is asked to type in exactly this text.
This text consists of 13 German words (94 characters without spaces, 106 with spaces):

Dialyse ist ein Blutreiningungsverfahren, das bei
Nierenversagen als Ersatzverfahren zum Einsatz kommt.

This text has to be inputted to the handwriting dialog and to the keyboard dialog in the experiment phase.

At the end of the experiment, the user has to fill out a feedback formulary, giving information about his subjective feelings about the system.

Figure 45: Participant during experiment in real life

5. Results

5.1. Optimized Timeout

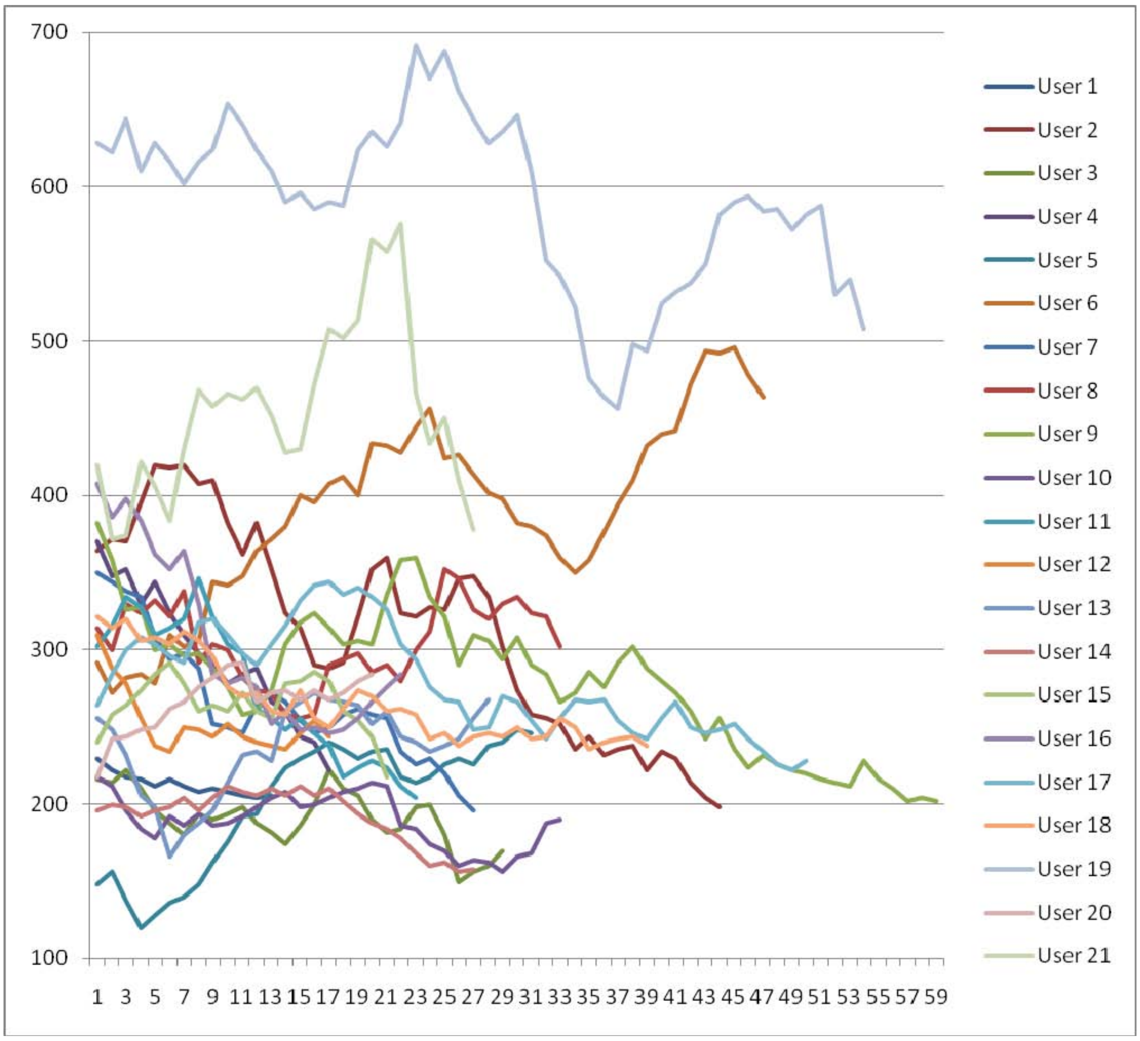


Figure 46: Adaptation of the timeout for each user

Figure 46 shows the development of the timeout for each user during the experiment. The values on the y-axe are the optimized timeout for each user, the x-axe shows the number of changing optimized timeout.

The figure shows, that the timeout for most of the users is between 200 and 250. The algorithm adapts the timeout to slow writers and for fast writers, which is shown by high timeout of one of the users.

Also adaptations to each user during writing are done, the figure shows that, for example, on User 19 the writing speed increases and decreases – just the way the user is writing.

5.2. Accuracy

Table 13: Accuracy with virtual keyboard [%]

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
99.1	6.28	100	11.5	99.06	1.44

Table 14: Recognition accuracy of handwriting recognition [%] with interventions

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
89.25	34.3	91.43	30.20	88.00	37.34

Table 15: Recognition accuracy of handwriting recognition [%]

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
84.66	57.6	86.99	79.15	83.33	38.21

The 85 year old participants have an accuracy of 89.2% for inputting text with the virtual keyboard and a recognition accuracy of 80.1% with interventions and 65.6% without interventions.

The 68 year old participants have an accuracy of 100% for inputting text with the virtual keyboard and recognition accuracy of 95% with interventions and 90.8% without interventions.

5.3. Writing Speed

For the calculation of the accurate words per minute, the following formula is used:

Equation 7: Accurate words per minute

$$accurate\_wpm = \frac{wpm * accuracy}{100}$$

Table 16: Writing speed in words per minute for virtual keyboard

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
13.17	27.7	12.88	29.46	13.43	18.29

Table 17: Writing speed in accurate words per minute for virtual keyboard

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
13.00	27.43	12.88	30.52	13.10	18.63

Table 18: Writing speed in words per minute for handwriting recognition

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
8.44	4.59	8.11	5.37	8.71	1.95

Table 19: Writing speed in accurate words per minute for handwriting recognition

Overall		≤ 30 weekly usage		> 30 weekly usage	
Median	Variance	Median	Variance	Median	Variance
7.47	4.56	7.43	6.47	7.75	2.05

The 85 year old participants wrote 2.87 words per minute with the keyboard and 2.82 wpm with handwriting recognition.
The 68 year old participants wrote 4.88 wpm with the keyboard and 4.17 wpm with handwriting recognition.

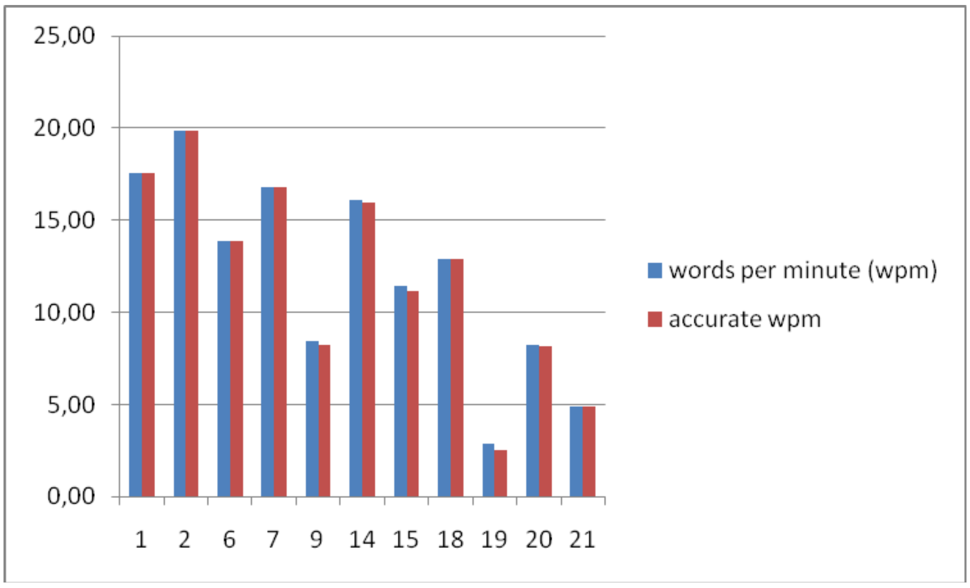


Figure 47: Speed and accuracy of keyboard input for participants using computer ≤ 30 hours

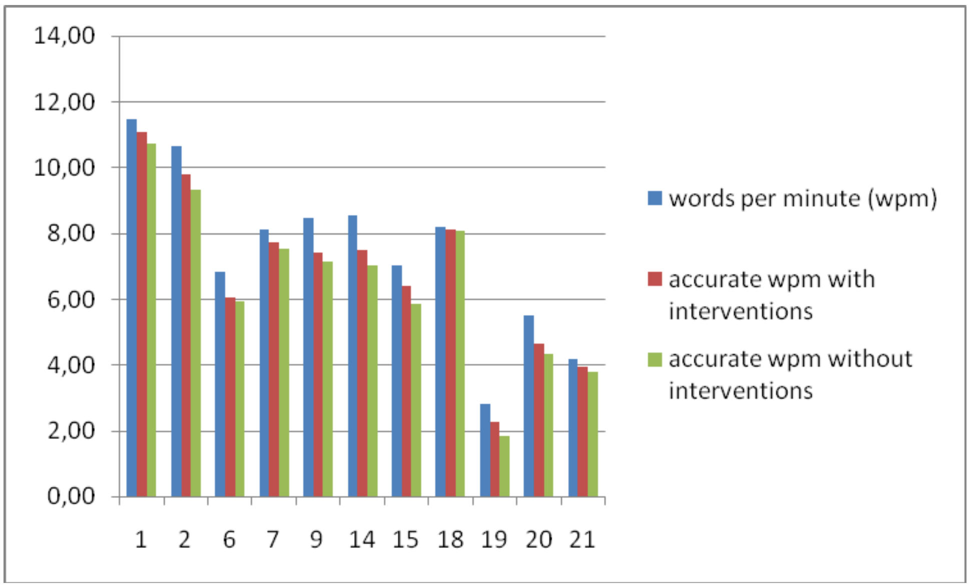


Figure 48: Speed and accuracy of handwriting recognition for participants using computer ≤ 30 hours a week

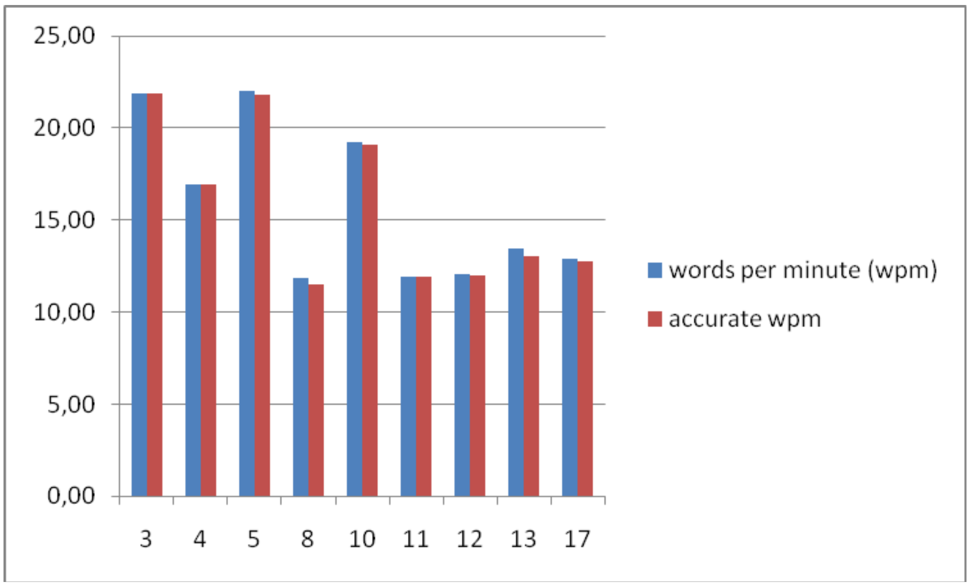


Figure 49: Speed and accuracy of keyboard input for participants using computer > 30 hours

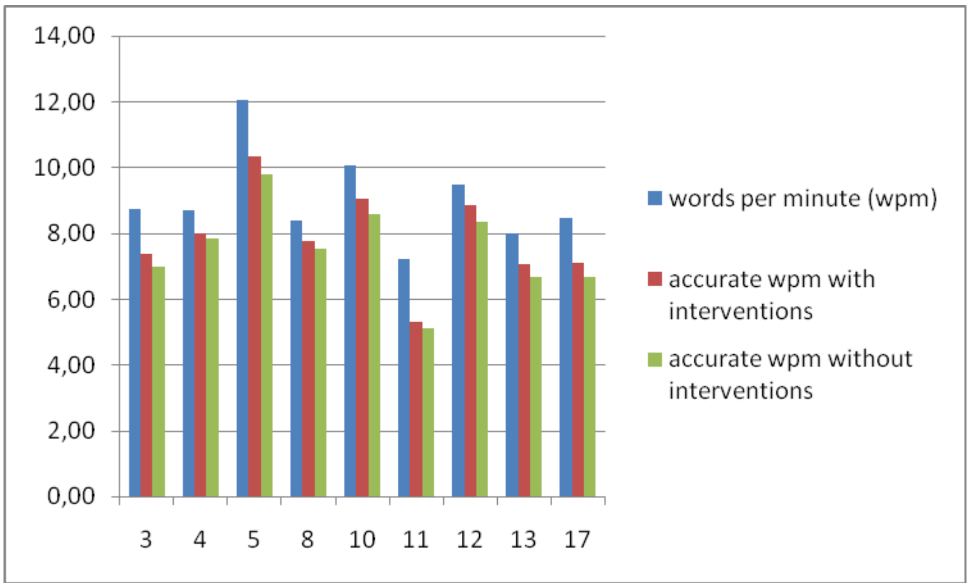


Figure 50: Speed and accuracy of handwriting recognition for participants using computer > 30 hours a week

5.4. User Questionnaire

Table 20: Overall results of user questionnaire

Overall	Median	Var
Keyboard		
Inputting Data (+4=easy, -4=difficult)	3.0	2.6
Correction of wrong inputted data (+4=easy, -4=difficult)	4.0	2.8
Handwriting		
Inputting Data (+4=easy, -4=difficult)	2.0	4.9
Correction of wrongly inputted/recognized data (+4=easy, -4=difficult)	3.0	1.4
Did the recognition slow down your writing (+4=no, -4=yes)	0.5	9.2
I would prefer (+4=handwriting, -4=keyboard)	-2.5	7.9
Basic Information		
Use of colour is (+4=useful, -4=useless)	2.0	2.5
The handwriting recognition positively surprised me (+4=yes, -4=no)	2.5	7.1
Characters on the PDA are easy to read (+4=yes, 4=no)	4.0	3.9

Table 21: Results of user questionnaire for weekly usage of computer ≤ 30 hours and above

Weekly computer usage [hours]	Median (<=30)	Var (<=30)	Median (>30)	Var (>30)
Keyboard				
Inputting Data	3.5	1.4	3.0	3.7
Correction of wrongly inputted data	4.0	1.7	3.5	4.5
Handwriting				
Inputting Data	2.5	4.2	0.5	3.8
Correction of wrongly inputted/recognized data	4.0	0.5	2.5	1.8
Did the recognition slow down your writing	2.5	8.4	-0.5	8.2
I would prefer	-1.5	2.0	-2.5	4.9
Basic Information				
Use of colour	2.0	2.5	1.5	2.6
The handwriting recognition positively surprised me	4.0	7.7	0.0	4.7
Characters on the PDA are easy to read	4.0	4.6	4.0	3.2

6. Discussion

Writing a text with a virtual keyboard by tipping with a stylus on the virtual keys of the keyboard gives an overall median accuracy of 99.1%, variance 6.28. As written in (Holzinger et al., 2008), the use of a stylus is very accurate. The speed was overall median of 13.17 words, variance 27.7. (Table 16, Table 13)
Participants using a computer more than 30 hours a week write faster than participants with computer usage ≤ 30, but surprisingly there are also more errors (Table 16, Table 13, Figure 47 Figure 49). Six of 9 participants of the group using a computer for more than 30 hours a week made at least one mistake, whereas 5 of 11 made at least one mistake in the other group.

Compared to the virtual keyboard, the speed and the accuracy of the handwriting recognition are lower. The overall median accuracy was 89.25%, variance 34.3 (Table 14). This does not reach 97%, which accuracy will be accepted by the user (LaLomia, 1994).
The overall median speed was 8.44, variance 4.59, which is nearly 5 words per minute slower than the virtual keyboard input. (Table 18)
Focusing once again on the two groups (usage of computer ≤ 30 hours a week and above), the accuracy of the group ≤ 30 is median 91.43, variance 30.20 whereas the median accuracy of the group > 30 is 88.00, variance 37.34. This is a difference of more than 3%. (Table 14)
The group > 30 was writing faster as the group ≤ 30 (Median 8.71 wpm, variance 5.37; median 8.1 wpm, variance 1.95).
Figure 48 and Figure 50 show the speed and accuracy of the handwriting recognition. The gap between the measured wpm and the accurate wpm is bigger for the group > 30 than for the other group.
The median values of the accurate words per minute nearly give the same writing speed for both groups (Table 19).

Maybe, people using a computer very frequently (more than 30 hours a week) get more and more used to write also e.g. notes on the computer instead of taking notes on a sheet of paper. There are hardly any letters written in handwriting. Maybe there is a link between more frequent usage of computers and writing faster, but handwriting more illegible or unreadable.

There were hardly any problems with data acquisition via virtual keyboard; the background questionnaire shows this with a median value of 3.0, variance 2.6. Also correction of wrongly inputted characters was very easy for the participants. (Table 20)

The handwriting recognition was rated to be rather easy by the group ≤ 30 (Median 2.5, variance 4.2), whereas it was rated neutrally by the other group (Median 0.5, variance 3.8). The correction of wrongly input data was rated to be very easy by the group ≤ 30 (Median 4.0, variance 0.5) and to be rather easy by the other group (Median 2.5, variance 1.8). (Table 21)

Generally, the participants would prefer the keyboard input (Median -2.5, Variance 7.9) but they were positively surprised by the handwriting recognition (Median 2.5, Variance 7.1). (Table 20)

The data acquisition via handwriting recognition was rated to be rather hard (Median 3.9, Variance 6.6) and the data input via keyboard to be quite easy (Median 7.2, Variance 1.8) in (Chittaro et al., 2007), on a scale where 0 means hard and 9 easy. Our results (4 to -4 scale) transformed to the other scale show the improvement in data acquisition with handwriting recognition.

Table 22: Comparison between Chittaro study and our study

	Chittaro Study		Our Study	
	Median	Var	Median	Var
Writing text using handwriting (0=hard, 9=easy)	3.8	6.6	6.5	5.19
Writing text using virtual keyboard (0=hard, 9=easy)	7.2	1.8	7.5	2.69

One participant was over the 97% accuracy gap for acceptance of handwriting recognition with an accuracy of 99.22%. Anyway, the user felt that the handwriting recognition would slow him down (-4) and definitely would prefer the virtual keyboard (-4).

There were no problems or confusions about the dialogs or the data acquisition, neither in the case of handwriting recognition nor in the case of the virtual keyboard. Corrections of wrongly recognized or inputted characters caused no problems to the user.

The two elderly people were included in this study in order to obtain data concerning participants who never used a computer or a mobile device before. For these two participants, there is hardly any difference between the input speed via virtual keyboard and handwriting recognition (Figure 47, Figure 48; User 18 and 21).
The handwriting recognition accuracy of the 68 year old participant was 95%, compared to median accuracy; it is a very high value.
The difference in speed between handwriting and the virtual keyboard for the participants (except the two elderly people) is very high, while for the elderly people the speed is nearly the same.
The results of these participants are interesting; however, it is not of practical relevance, since there are hardly any people left – at least amongst people able to volunteer as a first responder – without experience on computer keyboards. Today, children get used to work with computers, at home playing computer games, surfing the internet or at school.

7. Conclusion

Compared to the “Ambulance Run Reporting” study (Chittaro et al., 2007), our developments improved the usability of acquiring data with handwriting recognition.

Calibration, Continuous Calibration and Correction Intervention as well as other interventions on results delivered by the handwriting recognition engine achieved improvements of nearly 5% in recognition accuracy.

Because of choosing character recognition instead of word recognition, we were able to improve the accuracy and the usability of correction of recognition errors.

The optimization of the timeout creates an adaptation to the user, so that slow handwriting and fast handwriting is supported. The statistics recorded during the experiment show the steady adaptation (Figure 46), but it was rated very neutral by the participants in the feedback questionnaire (Table 20: Overall results of user questionnaire).

The user questionnaire at the end of the experiment, asking about the subjective feeling of the user about the handwriting recognition shows the improvement of the usability on acquiring data with handwriting recognition.

8. Future Work

Although much research in the field of handwriting recognition has been done, recognition algorithms do not achieve the high prospects of the users.
Handwriting is very individual to every person and identifying characters for humans is still very hard (Neisser and Weene, 1960).

This thesis showed operations on the result of a recognition engine. Replacing the engine used in this thesis' experiment (Calligrapher) with a better recognition engine with higher accuracy can improve the result on accuracy.
Because of typing in characters one by one, a word completion feature could be added to the final implementation. Less is more (Holzinger and Errath, 2007).

In the last years, mobile devices and the usage of such devices changed very fast. Nowadays, many people, especially younger people are connected to social networks like Facebook not only with their laptop or PC, but also with their mobile phones – smart phones.
This smart phones' user interface is a touch screen. Data acquisition is mostly realized with improved, intelligent virtual keyboards.
Intelligent keyboards e.g. with implementation of a regional error correction (Kwon et al., 2009) connected with tactile feedback for touch screen widgets (Koskinen et al., 2008) could improve performance and usability of virtual keyboards on small screens.

Handwriting is taught from elementary school on and nearly everyone learns handwriting at school. Therefore, handwriting recognition was a very important technology for input interfaces of mobile computers.
Nowadays, children get used to the QWERTY layout keyboard from elementary school on. Now, interface designers can assume that nearly everyone is experienced in using a QWERTY layout keyboard.

Because of the higher user acceptance of current uncomfortable virtual keyboards compared to handwriting recognition, future developments and projects should focus on data acquisition based on intelligent, comfortable virtual keyboards.

9. Final Implementation

9.1. Development Environment

The development environment was the eMbedded version of Microsoft Visual C++. The Microsoft® eMbedded Visual C++ 4.0 tool is a desktop environment for creating applications and components for Windows CE. The programming language is C++. (Microsoft, 2010)

9.2. Specifications

The final implementation was designed together with Manfred Unger, following Object Orientated Design methods (Herold et al., 2005) and guidelines of FERK Systems.

Basically, design guidelines of FERK Systems were higher prioritised than clean object orientated design, because the final implementation should be designed and implemented for easy and fast adaptations by developers at FERK Systems.

The code has to be written in coding standard of FERK Systems (FERK-Systems, 2007, FERK-Systems, 2005).
To generate a documentation of the implementation, the code is documented with doxygen as well (Sourceforge, 2010a).

Basically, the final implementation should be a library which can be included in a project and an input dialog of the library is called by just one line of code.

9.3. Design and Implementation

Figure 51 shows the UML Design diagram.

The cores of the class diagram are the two dialog classes, CFHandwritingDialog and CFHandwritingCalibrationDialog, which are both subclasses of CDialog, the CFWritePanel and the Singleton CFPersonData.

9.3.1. CFHandwritingDialog

This dialog class consists of the input field (CFEdit), the CFButtonControlBar which places the control bar buttons (Keyboard, Space, Backspace, New Line and UpperCase). The other buttons for En/Unroll and OK and Cancel are direct members of the class.
Also the Write Area (CFWritePanel) is a member of this class. The functionality of the ONCLICK events of all the buttons, also the buttons of CFButtonControlBar are implemented in this class.
Buttons are type of class CFBitmapButton.

The interface, to call this dialog is represented as a static method

```
static int DoDialog(CWnd *apParentWnd, int aDialogType, CString aCaption,
CString *aData, CString aPersonData)
```

- ***apParentWnd:** Pointer to the parent window
- **aDialogType:** Type of dialog
    - DIALOG_TYPE_TEXT
    - DIALOG_TYPE_MULTILINE_TEXT
    - DIALOG_TYPE_NUMBER
    - DIALOG_TYPE_DATE
    - DIALOG_TYPE_TIME
- **aCaption:** The caption of the dialog
- **aData:** Data for the input field. The result is also stored to this parameter
- **aPersonData:** The filename of the Person's user profile

This method will initialize and show the dialog.

9.3.1.1. CFHandwritngCalibrationDialog

This dialog class consists of two buttons (OK, Cancel, type of class CFBitmapButton) and the writing is (CFWritePanel).
Therefore, also the workflow of the calibration is implemented in this class; an object of the class CFCalibrationRecognitionHandler is a class member, too.

The interface to call this dialog is similar to CFHandwritingDialog a static method:

```
static int DoDialog(CWnd *apParentWnd, CString *aData, CString aPersonData)
```

- ***apParentWnd:** Pointer to the parent window
- **aPersonData:** The filename of the Person's user profile

9.3.2. CWritePanel

CWritePanel handles all the functionality belonging to handwriting.
This class displays the writing area on the screen. When the user is writing within this writing area, the class draws the writing to the screen. After recognition, the writing area will be cleaned of the writing.

The drawn handwriting is also stored in this class. TBS is measured here, too and the optimized timeout is calculated here.
The call of the handwriting recognition thread is in this class. After the handwriting recognition has finished, the results can be retrieved in the method OnRecResult.

In OnRecResult, the retrieved data is interpreted.
In case of the CFHandwritingDialog, the method InterpretResult (static method in CFFunction) interprets the result of the handwriting recognition (Chapter 3.10, Correction Intervention). After that other inventions are made to the results (Chapter 3.13, Other Interventions on recognition results).
After this step, the result is sent via keyboard event to the Text field on the CFHandwritingDialog. We used keyboard events, because the handling with, for example, marked text is done by the operating system.
The retrieved data is also stored to CFPersonData for continuous calibration (3.12Continuous Calibration).

If the parent dialog is CFHandwritingCalibrationDialog, the OnRecResult handles the result for Calibration (Chapter 3.11, Calibration) with the help of CFCalibrationRecognitionHandler.

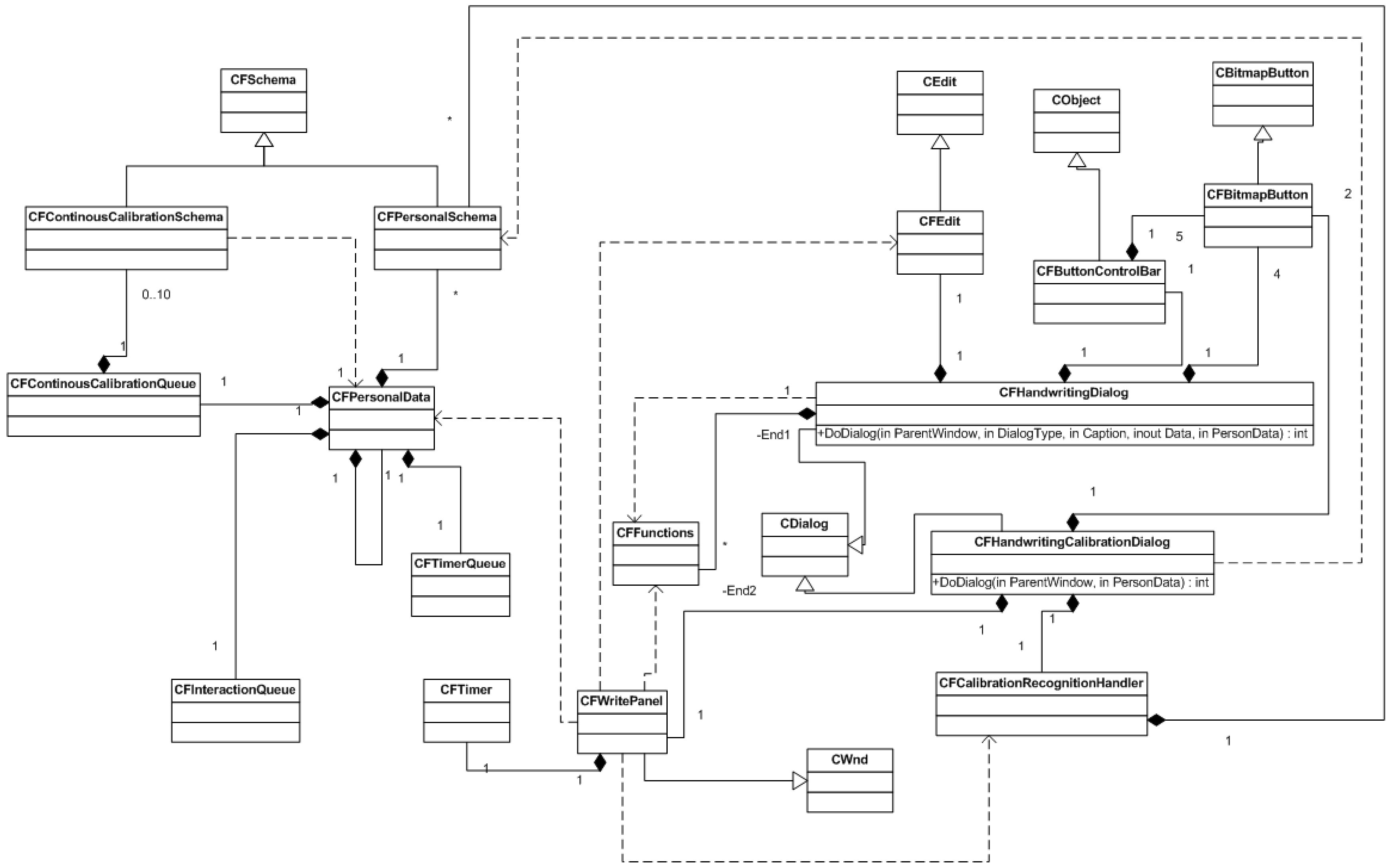
9.3.3. CFPersonalData

This class, implemented as Singleton (Herold et al., 2005) and contains the personal data of a user, which contains

- the queue of TBS,
- for each letter the weight of correctly recognized letters
- schemes of wrongly recognized letters
- temporary data for continuous calibration

Calling the GetInstance() method of the singleton with the filename of the user-profile-file will load the user-profile to the Singleton.

On closing the dialog, the Singleton object will be destroyed, but before this is done, the user data is stored back to the user-profile-file.



CFSchema
CFContinousCalibrationSchema
CFPersonalSchema
CFContinousCalibrationQueue
CFPersonalData
CFTimerQueue
CFInteractionQueue
CFTimer
CFWritePanel
CFFunctions
CDialog
CWnd
CEdit
CFEdit
CObject
CFButtonControlBar
CBitmapButton
CFBitmapButton
CFHandwritingDialog
+DoDialog(in ParentWindow, in DialogType, in Caption, inout Data, in PersonData) : int
CFHandwritingCalibrationDialog
+DoDialog(in ParentWindow, in PersonData) : int
CFCalibrationRecognitionHandler
-End1
-End2
0..10
*
1
2
4
5

Figure 51: Class diagram of the final implementation

9.4. Usage of the Library

The interface of the library is defined by two method calls.

9.4.1. Handwriting Dialog

Static method call:

```
CFHandwritingDialog::DoDialog(ParentWindow, DialogType, Caption, Data,
PersonData)
```

- **ParentWindow:** Pointer to the parent window (e.g. this)
- **DialogType:** Decide which dialog is called
  - **Text Dialog:**
    CFHandwritingDialog::DIALOG_TYPE_TEXT
  - **Multiline Text Dialog:**
    CFHandwritingDialog::DIALOG_TYPE_MULTILINE_TEXT
  - **Number:**
    CFHandwritingDialog::DIALOG_TYPE_NUMBER
  - **Date:**
    CFHandwritingDialog::DIALOG_TYPE_DATE
  - **Time:**
    CFHandwritingDialog::DIALOG_TYPE_TIME
- **Caption:** The caption of the input field
- **Data:** Delivering of the initial data of the dialog. The resulting data is also written to this parameter
- **PersonData:** File name of the user-profile file

9.4.2. Calibration Dialog

Static method call:

```
CFHandwritingCalibrationDialog::DoDialog(ParentWindow, PersonData)
```

- **ParentWindow:** Pointer to the parent window (e.g. this)
- **PersonData:** File name of the user-profile file

9.5. Test Scenario

To ensure, that the functionality of the prototypes described and developed in Chapter 3 is correctly implemented in the final implementation, a test scenario was developed.
The main parts of the test scenario are testing…

- Basic dialog functionality (functionality of buttons and data input) of
    - Text dialog
    - Date dialog
    - Time dialog
    - Number dialog
- Correction Intervention (Chapter 3.10)
- Other Interventions (Chapter 3.13)
- Continuous Calibration (Chapter 3.12)
- Adaptive Optimized Timout (Chapter 3.9)

The final implementation includes writing statistical files for the test scenario. To check the correctness of the interventions, the originally recognized results of the handwriting recognition engine and the finally displayed letter of each interaction as well as the cause of the intervention is saved to statistic file.
This logging function is implemented in #ifdef blocks, writing statistic files can be enabled and disabled.

9.5.1. Terminology

Liggesmeyer defines Failure, Fault and Error as follows:

- **Failure (Fehlverhalten):** Failures only can be identified dynamically when using a product, identifying unexpected functionality of the product.
- **Fault, Defect (Fehler, Defekt):** Faults are static errors in the code of a program, for example an incorrect implementation of a method/function.
- **Error (Irrtum):** Errors are mistakes done by programmers if some methods/functions are used in a wrong way, and therefore results a incorrect result.

(Liggesmeyer, 2009)

9.5.2. External Test Case Document

For test cases, an external document is created, which archives the specifications standard test protocol document of FERK Systems. This specifications includes the following:

- **I. Headline of the test case (Überschrift des Testfalls):** This includes the title and a short description of the test case.
- **II. Description of the test case (Testfallbeschreibung):** This describes hypothesis, if the functions of the test scenario are implemented correctly.
- **III. Test progress (Testverlauf):** Describes the steps of the test scenario in detail.
- **IV. Resume (Resumee):** Describes the result of the test case. The hypothesis could archive correct, partly correct or not correct. Partly correct and not correct requires further explanation.

9.5.3. Stage 1: Calibration

The test scenario starts with the calibration dialog executing the process of calibration (Chapter 3.11). In this calibration phase, 0…n letters causing problems are identified.

9.5.4. Stage 2: Input of letters causing problems

For each of these 0…n letters, the user is asked by the system to input each letter 5 times to the text input dialog (for each there is an instruction and text input dialog). In each step also a continuous calibration should be done in the background by the system.

9.5.5. Stage 3: Date input

In this stage, the correct functionality of the date input dialog is tested.
The user has to input the date "25.03.2010" and has to change it to "25.02.1983". The system should act like Table 23 shows.

Table 23: Interaction by the user and the result shown in the input field inputting a date

Input field	Inputted number or interaction
|TT.MM.JJJJ	2
2|T.MM.JJJJ	5
25.|MM.JJJJ	0
25.0|M.JJJJ	3
25.03.|JJJJ	2
25.03.2|JJJ	0
25.03.20|JJ	1
25.03.201|J	0
25.03.2010|	⇦
25.03.201|J	⇦
25.03.20|JJ	⇦
25.03.2|JJJ	⇦
25.03.|JJJJ	⇦
25.0|M.JJJJ	2
25.02.|JJJJ	1
25.02.1|JJJ	9
25.02.19|JJ	8
25.02.198|J	3
25.02.1983|	

9.5.6. State 4: Time Input

Also the correct functionality of the time dialog is tested.

The user has to input the time “10:19” and has to change it to “11:11”. The system should act like Table 24 shows.

Table 24: Interaction by the user and the result shown in the input field inputting a time

Input field	Inputted number or interaction
|SS:MM	1
1|S:MM	0
10:|MM	1
10:1|M	9
10:19|	⇦
10:1|M	⇦
10:|MM	⇦
1|S:MM	1
11:|MM	1
11:1|M	1
11:11|	

9.5.7. Stage 5: Text Input

The text input scenario tries to challenge problems with confusing pairs (Frankish et al., 1995) (e.g. “r” and “v”, “O” and “0”). The given text has to be inputted exactly:

```
Zu Ostern: ↵
01 Allegro↵
02 Andante↵
03 Scherzo↵
04 Allegro Vivace
```

9.5.8. Stage 6: Input of letters causing problems

Scenario from Stage 2 is repeated.

10.List of Equation

11.List of Figures

12.List of Tables

13.References

BAUMGART, D. C. 2005. Personal digital assistants in health care: experienced clinicians in the palm of your hand? *The Lancet,* 366**,** 1210-1222.

BOEHM, B. 1986. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes,* 11**,** 14-24.

BUNKE, H., ROTH, M. & SCHUKAT-TALAMAZZINI, E. G. 1995. Off-line cursive handwriting recognition using hidden markov models. *Pattern Recognition,* 28**,** 1399-1413.

CHANG, L. & MACKENZIE, I. S. 1994. A comparison of two handwriting recognizers for pen-based computers. *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research.* Toronto, Ontario, Canada: IBM Press.

CHITTARO, L. 2006. Visualizing Information on Mobile Devices. *Computer,* 39**,** 40-45.

CHITTARO, L., ZULIANI, F. & CARCHIETTI, E. 2007. Mobile Devices in Emergency Medical Services: User Evaluation of a PDA-Based Interface for Ambulance Run Reporting.

FERK-SYSTEMS 2005. Programmierrichtlinien für C/C++.

FERK-SYSTEMS 2007. Programmierrichtlinien Delphi 6.

FRANKISH, C., HULL, R. & MORGAN, P. 1995. Recognition accuracy and user acceptance of pen interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems.* Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co.

GADER, P. D., KELLER, J. M., KRISHNAPURAM, R., CHIANG, J.-H. & MOHAMED, M. A. 1997. Neural and Fuzzy Methods in Handwriting Recognition. *Computer,* 30**,** 79-86.

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.

GRONER, G. F. 1966. Real-time recognition of handprinted text. *Proceedings of the November 7-10, 1966, fall joint computer conference.* San Francisco, California: ACM.

GUERFALI, W. & PLAMONDON, R. 1993. Normalizing and restoring on-line handwriting. *Pattern Recognition,* 26**,** 419-431.

HALLER, G., HALLER, D. M., COURVOISIER, D. S. & LOVIS, C. 2009. Handheld vs. Laptop Computers for Electronic Data Collection in Clinical Research: A Crossover Randomized Trial. *Journal of the American Medical Informatics Association,* 16**,** 651-659.

HEROLD, H., KLAR, M. & KLAR, S. 2005. *C++, UML und Design Patterns*, Addison Wesley Verlag.

HOLZINGER, A. 2004. Rapid Prototyping for a Virtual Medical Campus Interface. *IEEE Softw.,* 21**,** 92-99.

HOLZINGER, A. 2005. Usability engineering methods for software developers. *Commun. ACM,* 48**,** 71-74.

HOLZINGER, A. & ERRATH, M. 2007. Mobile computer Web-application design in medicine: some research based guidelines. *Universal Access in the Information Society,* 6**,** 31-41.

HOLZINGER, A., GEIERHOFER, R., ACKERL, S. & SEARLE, G. 2005. The User Centered Development of a new Medical Image Viewer. *In:* ZARA, J. S., J. (ed.) *Central European Multimedia and Virtual Reality Conference.* Prague.

HOLZINGER, A., HÖLLER, M., SCHEDLBAUER, M. & URLESBERGER, B. 2008. An investigation of finger versus stylus input in medical scenarios. *In:* Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on, 2008. 433-438.

HOLZINGER, A., PEISCHL, B., DEBEVC, M. & SCHLÖGL, M. 2010. Preferences of Handwriting Recognition on Mobile Information Systems: Improving handwriting algorithm on the basis of real-life usability research. *In:* Proceedings of the International Conference on E-Business, 2010 Athens.

HOLZMAN, T. G. 1999. Computer-human interface solutions for emergency medical care. *interactions,* 6**,** 13-24.

KJELDSKOV, J., SKOV, M. B., ALS, B. S. & HØEGH, R. T. 2004. Is It Worth the Hassle? Exploring the Added Value of Evaluating the Usability of Context-Aware Mobile Systems in the Field.

KLANN, M., MALIZIA, A., CHITTARO, L., CUEVAS, I. A. & LEVIALDI, S. 2008. Hci for emergencies. *CHI '08 extended abstracts on Human factors in computing systems.* Florence, Italy: ACM.

KÖLSCH, M. & TURK, M. 2002. Keyboards without Keyboards: A Survey of Virtual Keyboards. Santa Barbara: Proceedings of Sensing and Input for Media-centric Systems.

KOSKINEN, E., KAARESOJA, T. & LAITINEN, P. 2008. Feel-good touch: finding the most pleasant tactile feedback for a mobile touch screen button. *Proceedings of the 10th international conference on Multimodal interfaces.* Chania, Crete, Greece: ACM.

KWON, S., LEE, D. & CHUNG, M. K. 2009. Effect of key size and activation area on the performance of a regional error correction method in a touch-screen QWERTY keyboard. *International Journal of Industrial Ergonomics,* 39**,** 888-893.

LALOMIA, M. 1994. User acceptance of handwritten recognition accuracy. *Conference companion on Human factors in computing systems.* Boston, Massachusetts, United States: ACM.

LEWIS, J. R. 1999. Input rates and user preference for three small-screen input methods: Standard keyboard, predictive keyboard, and handwriting. *Proceedings of the Human Factors and Ergonomics Society,* 43.

LIGGESMEYER, P. 2009. Einführung. *Software-Qualität.* Spektrum Akademischer Verlag.

MACKENZIE, I. S. & CHANG, L. 1999. A performance comparison of two handwriting recognizers. *Interacting with Computers,* 11**,** 283-297.

MACKENZIE, I. S., NONNECKE, B., RIDDERSMA, S., MCQUEEN, C. & MELTZ, M. 1994. Alphanumeric entry on pen-based computers. *International Journal of Human-Computer Studies,* 41**,** 775-792.

MACKENZIE, I. S. & SOUKOREFF, R. W. 2002. Text Entry for Mobile Computing: Models and Methods, Theory and Practice. *Human-Computer Interaction,* 17.

MADHVANATH, S., VIJAYASENAN, D. & KADIRESAN, T. M. 2007. LipiTk: a generic toolkit for online handwriting recognition. *ACM SIGGRAPH 2007 courses.* San Diego, California: ACM.

MANKOFF, J., ABOWD, G. D. & HUDSON, S. E. 2000. OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers & Graphics,* 24**,** 819-834.

MARTI, U.-V. & BUNKE, H. 2002. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition systems. *Hidden Markov models: applications in computer vision.* World Scientific Publishing Co., Inc.

MICROSOFT. 2010. *eMbedded Visual C++ 4.0* [Online]. Available: http://www.microsoft.com/downloads/details.aspx?FamilyId=1DACDB3D-50D1-41B2-A107-FA75AE960856&displaylang=en [Accessed 2010-05-05 2010].

NEISSER, U. & WEENE, P. 1960. A note on human recognition of hand-printed characters. *Information and Control,* 3**,** 191-196.

NOKIA Nokia 6120 Navigator.

NOYES, J. 1983. The QWERTY keyboard: a review. *International Journal of Man-Machine Studies,* 18**,** 265-281.

NOYES, J. 1998. QWERTY-the immortal keyboard. *Computing & Control Engineering Journal,* 9**,** 117-122.

PASCOE, J., RYAN, N. & MORSE, D. 2000. Using while moving: HCI issues in fieldwork environments. *ACM Trans. Comput.-Hum. Interact.,* 7**,** 417-437.

PHATWARE 2002. Calligrapher SDK 6.0 Developer's Manual.

PLAMONDON, R. & SRIHARI, S. N. 2000. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Trans. Pattern Anal. Mach. Intell.,* 22**,** 63-84.

READ, J. C. 2007. A study of the usability of handwriting recognition for text entry by children. *Interacting with Computers,* 19**,** 57-69.

READ, J. C., MACFARLANE, S. & GREGORY, P. 2004. Requirements for the design of a handwriting recognition based writing interface for children. *Proceedings of* *the 2004 conference on Interaction design and children: building a community.* Maryland: ACM.

ROYCE, W. W. Year. Managing the development of large software systems: concepts and techniques. *In:* Proc. IEEE WESTCON, 1970. IEEE Press.

SCHOMAKER, L. Year. User-interface aspects in recognizing connected-cursive handwriting. *In:* Handwriting and Pen-Based Input, IEE Colloquium on, 1994 1994. 8/1-8/3.

SHI, B. & LI, G. 2002. *VLSI neural fuzzy classifier for handwriting recognition*. United States patent application 10/064423

SOURCEFORGE. 2010a. *Doxygen* [Online]. Available: http://www.doxygen.org/ [Accessed 2010-05-10].

SOURCEFORGE. 2010b. *LipiTK* [Online]. Sourceforge. Available: http://lipitk.sourceforge.net/ [Accessed 2010-04-26 2010].

STRENGE, M. 2005. *Konzepte und Toolkits zur Handschrifterkennung.* Universität Leipzig.

TAPPERT, C. C., SUEN, C. Y. & WAKAHARA, T. 1990. The State of the Art in Online Handwriting Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.,* 12**,** 787-808.

XUE, H. & GOVINDARAJU, V. 2006. Hidden Markov Models Combining Discrete Symbols and Continuous Attributes in Handwriting Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.,* 28**,** 458-462.

YAEGER, L., WEBB, B. & LYON, R. 1998. Combining Neural Networks and Context-Driven Search for Online, Printed Handwriting Recognition in the Newton.

ZWICK, C., SCHMITZ, B. & KÜHL, K. 2005. *Designing for Small Screens*, AVA Publishing.

14. Commented References

CHANG, L. & MACKENZIE, I. S. 1994. A comparison of two handwriting recognizers for pen-based computers. *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research.* Toronto, Ontario, Canada: IBM Press.

This study compares two commercial handwriting recognizers (Microsoft Pen for Windows, CIC Handwriter) on a 640x480 LCD screen, testing each recognizer in Lowercase and Upper-Lower case mode.
16 volunteer subjects attended the experiment, entering characters provided by the software in four sessions. Each session contained entering data for each of the four conditions (each recognizer Lowercase and Upper-Lower case mode).
Due to learning, subjects increases in writing speed in the four sessions, while there was no significant effect on recognition accuracy.
For each recognizer, the lowercase mode yield to better recognition result.

CHITTARO, L., ZULIANI, F. & CARCHIETTI, E. 2007. Mobile Devices in Emergency Medical Services: User Evaluation of a PDA-Based Interface for Ambulance Run Reporting.

This study describes the design and evaluation of an easy-to-use mobile system for Ambulance Run Reporting.
The traditional run sheet for reporting a case of emergency is implemented to a PDA based mobile system. The data acquisition was done via the virtual keyboard and handwriting recognition (Calligrapher).
Part of the experiment was also an user questionnaire, where the participants rated data acquisition with the virtual keyboard much better than input with handwriting recognition. The space for writing was too small and most of the entered data was not recognized correctly. The handwriting recognition figured out a usability problem.

The weak point of the user evaluation is the number of participants, because only 6 participants (first responders) were attending the evaluation.

LALOMIA, M. 1994. User acceptance of handwritten recognition accuracy. *Conference companion on Human factors in computing systems.* Boston, Massachusetts, United States: ACM.

This study tries to find out how high handwriting recognition accuracy has to be to find it useful for people.
Participants were writing specified text to a system, which immediately interprets the input with randomly occurring errors. The randomly generated accuracy ranged from 90% to 99%. After reviewing the errors, the participants rated acceptability.
Recognition accuracy over 97% was acceptable.