A study on creating a custom South Sotho spellchecking and correcting software desktop application

Leon A. Grobbelaar
N.Dip., B.Tech.

Study submitted in accordance with the requirements of the degree

MAGISTER TECHNOLOGIAE: INFORMATION TECHNOLOGY

in the

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

of the

FACULTY OF ENGINEERING, INFORMATION AND COMMUNICATION TECHNOLOGY

at the

CENTRAL UNIVERSITY OF TECHNOLOGY, FREE STATE

2007

Supervisor: Prof. J.K. Kinyua

Declaration

I, Leon A. Grobbelaar, student number 20040253, do hereby declare that this research project which has been submitted to the Central University of Technology, Free State for the degree MAGISTER TECHNOLIGIAE: INFORMATION TECHNOLOGY, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology, Free State; and has not been submitted before by any other person in fulfillment, or partial fulfillment, of the requirements for the attainment of any qualification.

_____ _____

Signature of student Date

Acknowledgements

The author would like to thank the following entities and people for all their help, guidance, support and understanding:

- *The Good Lord; for providing me with the necessary insight at times when I faced new and difficult obstacles whilst busy with this study as well as for providing me with this opportunity to further my studies and honor Him through it.*
- *Professor J.K. Kinyua; for the hours spend assisting and mentoring me with regard to research methodologies, how to apply a chosen methodology successfully and how to write a satisfactory dissertation.*
- *Elriëtte Herbst; whom without I may have given up before I have completed the study. Your support and motivation really pulled me through when the work became very challenging.*
- *Gerrit, Christa, Gerrit and Barnie Herbst; the phone calls and messages of support inspired me to push on and complete the work at hand.*
- *My parents and family; for their unique brand of support and motivation, for believing in me, for the messages of encouragement and support as well as for all the assistance I have received from you.*
- *Bertram Haskins; for his input, guidance and assistance. It is much appreciated.*
- *Shane Hancke and my other friends; for your support and for indulging my somewhat anti-social behavior at times during the course of this study.*
- *The CUT and my superiors at the Welkom Campus; for the opportunity to engage in this endeavor as well as the understanding I have received.*

Preface

This has been one of the most interesting and involving periods of my life. The hard work, late nights and exploration is over, for now. The work has had its own bitter-sweet rewards: I have conquered an objective through a lot of sacrifice and perseverance. The feeling of satisfaction and victory is quite inexplicable, as I would imagine it to be for the man who has conquered Mount Kilimanjaro or Everest for the first time. It is here then, on my own mountain peak, that my long journey of discovery and, at times, mixed emotions of frustration, joy and victory, culminates.

"The journey of a thousand miles starts with one step."

Confucius

Summary

The researcher has created a multithreaded, spell checking and correcting software application for the Windows platform, called eSpellingPro sa Sesotho sa Leboa, specifically targeted to check South Sotho typed text for misspelled words. Accomplishing this goal meant that the system had to monitor and examine typed words, flag incorrectly spelled words and pass correctly spelled words as exactly that.

The biggest motivation behind the development of this system, a custom spell checking and correcting programme for the indigenous South African language, South Sotho and not just simply an add-on to existing systems, stemmed from evidence that has been gathered that suggested the average twenty year old South Sotho individual's spelling-skills have deteriorated. When considering the need to create error free documents, for example, a legal document in South Sotho, sub-standard spelling-skills could pose possible problems.

In addition to checking for misspelled words and flagging them, the system also has a degree of automatic error correction and suggests possible correct forms of spelling for the misspelled word to the user. Additionally, the system has the ability to translate South Sotho words into their Afrikaans and English equivalent meanings. Added to the former are simple features of existing spell-checkers, for example the ability to change the font, the font-size, to apply bold, italics or both to a word, underline a word, select all the text in the document and to print the document.

The application operates by capturing each word after the spacebar has been pressed. After the word has been captured, the application continues by analyzing the word, stripping off its prefix and checking it, checking the remainder of the word, that is, the characters of the word left after the prefix has been removed as well as checking the whole of the word, without stripping off the prefix. The word and the individual parts are checked according to their unique hash codes.

The pre-mentioned all occur simultaneously by assigning each operation to its own processing thread. There is no need to extract a suffix from a word, because suffixes do not form part of the South Sotho vernacular. For example, in English one would use the word "*acres*" to suggest more than one acre of land, "*s*" suggesting the plural, while in South Sotho one would use "*diakere*", "*di*" suggesting the plural form.

After the checking algorithms have been executed, the application makes suggestions about the misspelled words based on the individual similarity keys, calculated by the software, and fills the appropriate list box with these suggestions. The user then has the option to choose one of the suggestions made, after which the program replaces the misspelled word with the suggestion chosen by the user. The program has a degree of automatic correction ability, which allows it to automatically correct misspelled words.

Although South Sotho is not a language with high inflection, the application also checks for inflection, for example when the user enters two words as one. The system recognizes this and suggests two separate words to the user, next to each other as a sentence-portion as well as other alternatives spelled correctly.

To accomplish faster times vis-à-vis the checking, flagging and suggestion operations, certain modules were loaded into their own processing threads. This, so that the CPU's processing power could be better utilized, which resulted in more satisfactory checking, flagging and suggestion times.

With regards to scope, the application was somewhat limited. The dictionary only contained a lexicon set of two hundred and thirty three words, approximately nine words for each letter of the alphabet, where applicable. It only recognizes the full stop as a punctuation sign, it can only translate one word at a time instead of a sentence, but, as this study's objective was to create a spell-checker and -corrector, this was deemed acceptable. The researcher discusses possible solutions to these challenges in the final chapter.

In terms of user friendliness, the application was developed with controls easily recognizable to any user who has used a similar product in the past: Novice users, as with any software product, will need an amount of training. The application boasts controls, information and error messages all named and displayed in South Sotho, since the targeted user group is South Sotho speaking individuals.

A DVD was used to apply the system. The system was tested in two fold: Firstly the system was tested with regard to response times for the checking, flagging and suggestion operations, each tested independently and then as a whole. Secondly, the system was tested for correct- and incorrect flagging of misspelled (or correctly spelled) words. If a correctly spelled word is flagged as incorrect, it meant that a false negative flag occurred. If an incorrectly spelled word was flagged as incorrect, it meant that a true negative occurred. Two other types of flags could have occurred: A true positive, which meant that the spell checker and corrector identified valid words and did not flag them as incorrect, and then also a false positive. A false positive occurred when a word has not been identified by the program, resulting in a missed flag.

Overall, the system yielded more than acceptable results which will be displayed in chapter five. The success rate of the system meant that the system could be applied to a real-life environment.

Opsomming

Die navorser het ’n multi-draad spel-evaluasie en -korrigasie sagteware program vir die Windows platvorm, byname eSpellingPro sa Sesotho sa Leboa, ontwikkel. Die sagteware program is spesifiek daarop gemik om Suid-Sotho, getikte, dokumente te monitor en te ondersoek vir spelfoute. Om hierdie doelwit te bereik sou beteken dat die stelsel elke getikte woord moes evalueer, verkeerd gespelde woorde moes identifiseer en woorde wat nie verkeerd gespel is nie, nie as verkeerd gespel te identifiseer nie.

Die grootste motivering om hierdie tipe stelsel te ontwikkel, wat as eindproduk ’n unieke, doelgeboude, spel-evaluasie en -korrigering sagtewareprogram vir die inheemse Suid-Afrikaanse taal, Suid-Sotho, aangeneem het, was gebasseer op bewyse wat daarop gedui het dat die gemiddelde Suid-Sotho individu van ongeveer twintig jaar oud, se spelvermoeëns afneem of besig is om af te neem. Indien ‘n mens in ag neem dat daar ‘n behoefte bestaan om, byvoorbeeld, ‘n foutlose regsdokument in Suid Sotho te produseer en daar ook in ag geneem word dat spelvermoëns sub-standaard is, word dit duidelik dat daar ’n moontlike probleem in hierdie opsig bestaan.

Behalwe vir die bogenoemde spel-evaluasie- en -korrigeervermoeëns van die program, sluit dit ook ’n mate van outomatiese korrigering in en stel dit woorde aan die gebruiker voor as moontlike korrek gespelde woorde vir die woorde wat in die teks as verkeerd gespel, geidentifeiseer is. Die stelsel sluit ook die vermoë in om woorde van Suid-Sotho na Afrikaans en Engels te vertaal. Verder sluit die stelsel funksies in wat in menige ander spel-evaluasiesagteware as standaard voorkom, wat byvoorbeeld die vermoëns om die skrif en die skrifgrootte van woorde te verander, woorde in “dik-druk” te vertoon, woorde te onderstreep, om woorde in “skuins-druk” te vertoon of al van die voorafgenoemde toe te pas, insluit.

Die toepassingsagteware ontvang elke getikte woord nadat die spasie sleutel op die sleutelbord gedruk is. Nadat die woord ontvang is gaan die program voort om die

woord te analiseer deur die voorvoegsel van die woord te verwyder en dit na te gaan vir korrektheid. Dieselfde word met oorblyfsel van die woord, bedoelende die deel van die woord wat oorbly nadat die voorvoegsel verwyder is, asook die hele woord sonder dat die voorvoegsel verwyder word, gedoen. Die voorafgenoemde word gelyktydig deur die program uitgevoer deurdat die program die segmente wat die evaluering doen, elk in sy eie verwerkingsdraad laai en daardeur sorg dat die sentrale verwerker van die rekenaar beter benut word. Dit is nie nodig dat die program die agtervoegsel van Suid- Sotho woorde verwyder nie, omdat agtervoegsels nie deel van die Suid-Sotho taal vorm nie, byvoorbeeld, in Afrikaans sal die woord "*hektare*" die meervoed van "*hektaar*" aandui, waar die "*e*" die meervoud aandui. In Suid Sotho word die woord "*diakere*" as meervoud vir die woord "*akere*" gebruik, waar "*di*" die meervoudvorm voorstel.

Nadat die evaluasie en identifikasie algoritmes uitgevoer is, gaan die program voort om voorstelle vir die verkeerd gespelde woorde wat geidentifiseer is, te maak. Die program maak gebruik van eendersheids-sleutels wat deur die stelsel uitgewerk word vir die proses. Daarna lys die program die voorstelle, waarna die verbruiker van die voorstelle kan kies. Die program vervang dan die verkeerd gespelde woord met die voorstel wat deur die verbruiker gekies is. Die sagteware sluit ook 'n mate van outomatiese korrigering in, waarop 'n verkeerd gespelde woord deur die program geïdentifiseer en dan outomaties gekorrigeer word.

Alhoewel Suid-Sotho nie 'n taal is wat bekend is vir woordverbouings, waar twee woorde bymekaar gevoeg word om 'n nuwe woord te vorm nie, toets die program wel vir sulke gevalle. Dit mag gebeur dat die gebruiker van voorneme is om twee woorde te tik, maar per ongeluk die twee woorde as een tik. Die program analiseer die woord en stel dit as twee aparte woorde of die twee woorde langs mekaar as 'n sinsnede voor.

Om vinniger evaluerings- , identifikasie- en voorsteltye te vermag, word, soos voorheen genoem, sekere kodesegmente elk in hul eie verwerkingsdraad gelaai. Dit verseker dat die verwerkingshulpbronne gebruik, ten volle benut word. Laasgenoemde verseker ook

dat die program beter evaluerings- en voorsteltye behaal.

In terme van omvang, is die program redelik beperk. Die elektroniese woordeboek wat opgestel is, bevat slegs twee honderd drie en dertig woorde, ongeveer nege woorde vir elke letter in die alfabet, waar van toepassing. Dit herken slegs die punt as ‘n leesteken en kan op die oomblik slegs een woord vertaal na Afrikaans en Engels en nie ‘n sin nie. Die studie se hoofdoel was om ‘n spel-evaluasie en -korrigeerderprogram te ontwikkel, dus was die limitasies wat genoem is, as aanvaarbaar geag. Moontlike oplossings vir bogenoemde limitasies word in die finale hoofstuk kortliks bespreek.

Nadat gebruikersvriendelikheid in ag geneem is, is die program ontwikkel met die uitleg van bekende bestaande spel-evalueringssagteware, wat vir die gesoute gebruiker bekend is. Gebruikers wat nie bekend is met die tipe sagteware nie sal, soos met enige nuwe produk wat onbekend is, opleiding in die opsig moet ontvang. Die sagteware gebruikerskoppelvlak se kontroles is in Suid Sotho aangedui, so ook enige boodskappe wat aan die gebruiker gemaak word.

‘n DVD is as model vir die stelsel is gebruik. Die stelsel was in tweevoud getoets: Eerstens was die stelsel getoets in terme van terugvoertye vir die toetsing, identifisering en voorstelling van woorde. Elk van die toetse is afsonderlik en daarna as een tyds-toets gedoen. Tweedens is die stelsel getoets ten opsigte van die korrekte- en inkorrekte identfikasie van woorde wat verkeerd gespel is. Indien ‘n korrek gespelde woord as verkeerd geidentifiseer is, het dit beteken dat ‘n vals negatiewe identifikasie plaasgevind het. Indien ‘n verkeerd gespelde woord as verkeerd geïdentifiseer was, het dit beteken dat ‘n ware negatiewe identifikasie plaasgevind het. Twee ander tipes indentifikasie bestaan in die studie: ’n Ware positiewe identifikasie het plaasgevind wanneer die stelsel ‘n woord wat korrek gespel is, as korrek gespel aanvaar het en nie as verkeerd gespel geidentifiseer het nie. Dan bestaan daar ook ‘n geval van vals positiewe identifikasie, waar ‘n verkeerd gespelde woord nie as verkeerd gespel geïdentifiseer is nie.

Die stelsel het meer as aanvaarbare resultate getoon, wat in hoofstuk vyf bespreek word. Die sukses van die stelsel het beteken dat dit in ‘n bedryfsomgewing toegepas kan word.

Table of Contents

List of Figures

Chapter 3

Chapter 4

Chapter 5

Appendix A

Appendix B

List of Tables

List of Abbreviations and Definitions

Fn	False negative
Fp	False positive
Tn	True negative
Tp	True positive
OCR	Optical Character Recognition

Composition

a. the art of putting words and sentences together in accordance with the rules of grammar and rhetoric

Derivation - Grammar

the process or device of adding affixes to or changing the shape of a base, thereby assigning the result to a form class that may undergo further inflection or participate in different syntactic constructors, as in forming *service* from *serve*, *song* from *sing* and *hardness* from *hard* (contrasted with INFLECTION)

the process by which such a set of forms is derived

Inflection - Grammar

the process or device of adding affixes to the base word

the paradigm of a word

a single pattern of formation of a paradigm: noun inflection; verb inflection

the change in the shape of a word, generally by affixation, by means of which a change of meaning or relationship to some other word or group of words is

the affix added to produce this change, as the *-s* in *dogs* or the *-ed* in *played*

the systematic description of such processes in a given language, as in *serves* from *serve*, *sings* from *sing* and *harder* from *hard*

Linguistics

a. a set of forms, including the initial form, intermediate form and final form, showing the successive stages in the generation of a sentence as the rules of a generative grammar are applied to it
b. the process by which such a set of forms is derived

Morphological linguistics

the patterns of word formation in a particular language, including inflection, derivation and composition

the study and description of such patterns

the study of the behaviour and combination of morphemes

Chapter 1

1. INTRODUCTION

As South Africa has changed during the last decade, so too has the needs of its people. These range from seemingly simple needs such as day-to-day transport to more pressing issues such as electricity supply for the growing economy.

Policies have been amended to cater for every individual in the country and none as important as the changes to the rights of citizens regarding their language. South Africa has eleven official languages and these are now used in government documents, on national television and radio, and most importantly, education as well. All South African citizens can now receive their education in their native language if they wish.

When the “information age” or “computer age” dawned, so too did digital applications to enhance the quality of life, bringing easier, faster and more effective techniques to automate tedious tasks. For example, word processors like Microsoft Word have had a very profound impact, on a daily basis, on most businesses and form an integral part of many businesses. In addition, word processors and other productivity software applications are used widely by people in many different sectors of society. As the cost of computer technologies decreases and businesses grow, and the usage of computer technologies increases, so too will the need for a computer-based spell checking and correcting software.

Desktop processing power has increased exponentially over the last two decades, making way for faster and more complex word processors that use larger look-up dictionaries.

Many people are familiar with spell checking software such as Microsoft’s Word processor programme. This software initially supported spellchecking software for a

variety of languages including Czech, Dutch, English, German, Irish [24] [35] and so on. However, spellchecking support for some of our indigenous languages such as Sotho, Xhosa and Zulu has recently become available [30] [40] [41]. These additions have been created by institutions such as the North West University and can be purchased independently. The above mentioned additions to spellchecking software are operating system-specific, i.e. they were created as an addition to MS Word only.

Most operating systems have some representation of a spell checker in terms of a word processor of some sort and were developed for “mainstream” languages like English, French, German and Chinese. With the enhanced processing power and lower cost of desktop computers, it is possible to develop an automatic spellchecking and correcting application for a variety of indigenous South African languages to run in a desktop environment.

1.1 Statement of the problem

The role that computers play in document preparation has seen exponential increases. Applications such as word processors include functions such as spelling error detection [29] and correction [38], grammatical error detection and correction [6], triphone analysis [43], and so on, which play a large role when it comes to preparing error free documents. When it comes to dealing with issues and problems of language processing using computers, the area of computational linguistics should be investigated [7] [27].

Checking for spelling and typographical errors in computer-based texts, is a necessity. We can thus conclude that the area of computational linguistics, more specifically, checking for spelling errors, is a heterogeneous task, which might include obstacles like syntax, semantics, keyboard configuration, user profile, etc**.** [37].

In accordance with new government legislation, all South Africans should receive education in their mother language for at least the first five to seven years of their schooling by 2008 (the legislation is currently being implemented [1]). Furthermore, one

or more of the remaining eleven official languages (except Afrikaans and English) should be provided as an optional subject on secondary education level, if that language is not provided as a first language subject choice.

Many modern South African schools have a computer lab or facility on site and students are becoming increasingly proficient in the use of computers and computer technology. Although there is evidence that spellchecking software exists [35] [41] for the following official languages (excluding Afrikaans and English): SeSotho se Leboa (Northern Sotho), Setswana, Sesotho, isiXhosa, isiZulu, isiNdebele, SiSwati, Tshivenda and Xitsonga, no spell correcting software exists for South Sotho. If we consider the above-mentioned legislation and lack of specific software for our native languages (spoken by a large part of our population, more than 3.3 million [22]), a spelling corrector for a language like South Sotho would be a useful and innovative step forward and could form the basis for further exploration into creating spelling correcting software for the remainder of the languages [35].

Results based on statistical analysis (see chapter three) of a questionnaire which was completed by a student focus group, also contributed as motivation to undertake this project.

A software programme that can check words, make suggestions, correct misspelled words and give a word's equivalent, its synonyms, antonyms and its description as per dictionary in another language like English, could not only assist young learners in improving their basic spelling skills (when applied correctly) and in the exploration of their own language, but could also have business and industry applications in our metropolitan business and industry environment.

1.2 Motivation for the research

Why would we need a South Sotho spell checker and corrector? Although South Sotho might be perceived as a minority language on an international scale, more than 3.3 million South Africans speak it on a day-to-day basis [22]. Spell checkers/correctors form the basis of any document preparation as do the use of language in such a document.

If we couple this with the fact that the South African Government promotes and propagates education of South Africans in their home tongue in national legislation; it then becomes apparent why such an application is needed. It is also a scholarly belief that including a minority language in a spell checker/corrector leads to the survival and preservation of the language [6]. The less obvious advantages include a better understanding of other languages such as Afrikaans and English as well as enhancing the spelling skills of its users.

1.3 Study hypothesis

A spell checker and corrector application for the South Sotho language can be developed and run successfully and reliably on a personal desktop computing system, whilst:

i. Providing a satisfying ratio between spell-error identification and automatic correction of misspelled words.
ii. Performing reliably, that is, it should not flag correct words as incorrect nor should it observe incorrectly spelled words as correct.
iii. Providing the ability to make acceptable spelling suggestions for words that were identified as being incorrectly spelled.
iv. Encompassing the ability to translate a South Sotho word into Afrikaans and English.

v. Performing within set benchmarks with regards to checking, flagging, suggestion performance and accuracy.

For the hypothesis to be satisfied, the following criteria must be met:

i. The application must flag incorrectly spelled words as incorrect as well as not flag correct words as incorrect. Furthermore, incorrectly spelled words must not be observed as correctly spelled and correctly spelled words must be observed as being spelled correctly.
ii. The success rate of error identification (flagging) may be no less that 90% and the programme must employ a degree of automatic word-correction.
iii. The application must produce results consistently, that is, 75% of the time with no less than 75% suggestion accuracy.
iv. The application must perform flagging and suggestion within a time frame of 600 milliseconds respectively.
v. The application can constantly supply the correct Afrikaans and English translation of the identified South Sotho word per dictionary, whilst performing within the mentioned time frame.

The values of 90% for the success rate in terms of error identification, 75% for suggestion accuracy and a flagging time of 600 milliseconds were based on the findings of Van Huyssteen and Van Zaanen [40] with regard to their work. We did, however, make adjustments to the values Van Huyssteen and Van Zaanen [40] obtained e.g. our dictionary is smaller, so our application's suggestion time had to be faster when compared to the former.

1.4 Research goal and objectives

The aim of this study is to create a spell correcting application as a means to aid South Sotho speaking individuals with the identification or flagging of general spelling mistakes

made, providing suggestions to a high degree of efficiency as to the correct spelling for a misspelled word and where possible, to correct a mistake automatically.

To add functionality to this application, it will include the following as an objective related to the goal: It will incorporate the ability to translate South Sotho words into its Afrikaans and English equivalents.

1.5 Research methodology

Since the objective of this project is to develop a spell checker with automatic spell correcting capabilities, the core of this study was to develop the actual software application using Visual Basic.Net as the development language.

Purchasing an existing digital form of a South Sotho/English and a South Sotho/Afrikaans dictionary proved not only to be problematic, but also an expensive, task. Thus, making use of *Bukantswe Ya Maleme-Pedi* by Du Plessis, Gildenhuys and Moila [9], a digital dictionary was created by choosing a subset of words out of each respective letter in the alphabet in which a South Sotho word could be found (for example: no South Sotho word beginning with the letter "*c*" could be found in the mentioned dictionary).

Database tables were constructed that satisfied the criteria for this study and for the exclusive checking algorithms used. These are also discussed in Chapter five. All of this means that the study had limited scope, but without hampering performance in terms of efficiency, only in execution/flagging time, which, if we take processing power into account, would be in the micro-seconds territory.

Because of the unique nature of this project, benchmarking produced its own challenges. Regarding efficiency, the statistics produced by Van Huyssteen and Van Zaanen's (discussed briefly in Chapter three) [41] Afrikaans spell checker were used. Although the dictionary Van Huyssteen and Van Zaanen [40] used is much larger, it

does not affect efficiency due to the power of modern day processors. Regarding a benchmark for the time related to the flagging of errors: two checking-algorithms were tested and the results are documented in chapter five.

1.6 Organization of the dissertation

The organization of the rest of this dissertation is as follows.

Chapter 2 discusses the difference between a spell checker and a spell corrector. In *chapter 3* we shed light on a field survey that was done as well as quantifying the statistics that were extracted from this survey. *Chapter 4* follows with a short discussion of related work done in this field, whereas *chapter 5* focuses on *algorithm* design and implementation as well as the results obtained from these algorithms. The different algorithms' performances will also be discussed. In *chapter 6* the literature contains the admission of the conclusions reached for this study as well as future work that could be done and avenues for further work.

Chapter 2

2. SPELL CHECKER vs. SPELL CORRECTOR

This chapter is a more detailed review of the difference between a spell checker and a spell corrector. It also outlines where our intended software application fits in when considering its functionality.

2.1 A spell checker

According to thefreedictionary.com [24], the definition for a spell checker is "*An application within most word processing programs that checks for spelling errors in documents*" and according to Dictionary.com it is "*an electronic dictionary in a word processor that can be used to catch misspelled words*" [17] [20] [24].

Spell checkers do not incorporate any contextual knowledge [30] on their own, thus, the words will be checked with no contextual assumptions; nor in context. A spellchecking programme takes an input file of text and identifies words that are spelled incorrectly. Kukich [28] remains the authority concerning reference work [35]. She points out that three distinctions must be made between a spell checker and a spell corrector:

- error detection vs. error correction,
- interactive spelling checkers vs. *automatic spelling correction* and
- attention to isolated words vs. linguistic and textual context.

These form the basis of a spelling *checker*, especially the first two points. Spelling checkers are thus interactive programs that identify or *detect* spelling mistakes [31]. According to Silviu Cucerzan [33] of Microsoft, typical spell checkers compute, for each unknown word, a small set of in-lexicon alternatives

to be proposed as possible corrections, but very, very few attempts to detect *and correct* word substitution errors (which would make such a spell checker lean towards becoming a spelling corrector).

2.2 A spell corrector

According to Microsoft's Brill, a spelling corrector is able *to detect and correct* word substitution errors that result from typographical and even as far as cognitive mistakes [33]. This means a spelling corrector has the additional functionality of automatically correcting misspelled words with a high degree of accuracy over a spell checker. It also does not incorporate contextual knowledge with regard to error-identification of words within a text corpus. When there is research that aims to correct words in text, three increasingly more difficult problems become apparent regarding detection and automatic correction:

- non-word error detection
- context-dependent word correction and
- isolated-word error correction [28].

A spell corrector both detects incorrectly spelled words and tries to find the *most likely correct word with which to automatically replace the misspelled word* [6]. The possibility does exist, however, that the word with which the programme replaced the misspelled word with, was not what the user intended to type. Modern spell correctors can alert the user momentarily when an automatic correction is made.

Spelling correctors can also provide extra features like providing synonyms and antonyms for a word, providing plural forms of a word where applicable and words that have equivalent meaning in another language (e.g. an Afrikaans or English word for a specific South Sotho word). Desktop processing power has

increased exponentially over the last two decades, making way for more complex word processors that use larger look-up dictionaries, faster.

Spell checking software such as Microsoft's Word has become household names. This spell checking software catered for a variety of languages including Dutch, English, German, Irish etc. [24] [35] in the past, but it now even caters for some of our indigenous languages such as Sotho, Xhosa, Zulu and so forth [30] [40] [41]. All of these languages have found their way into application form. These additions have been created by institutions such as the North West University and can be purchased independently as add-on software. The above mentioned additions to spellchecking applications are operating system and application specific, i.e. it was created as an addition to MS Word only, running on the Windows platform.

Many operating systems have some representation of a spell checker in terms of a word processor of some sort and were developed for "mainstream" languages i.e. languages spoken by the section people with large buying power (English, French, German, Chinese) and especially languages spoken in developed countries.

2.3 eSpellingPro sa SeSotho sa Leboa

With the enhanced processing power and lower cost of desktop computers, it is possible to create a spellchecking and correcting application for a variety of indigenous South African languages to run on a desktop environment with ease, without restricting efficiency.

eSpellingPro sa SeSotho sa Leboa was the intended end result of this work. Our application program incorporates attributes of both spell checkers and spell correctors.

Typical spell correcting and checking software, like MS Word, boasts the following functionality:

- Identifies a misspelled word.
- Corrects the misspelled word automatically or supplies the user with a list of suggestions that might be considered by the user as being the correct word he/she intended to use, for example, the characters "*fg*" in text might yield the suggestions "*fig, figs, fog, fug, fig. (i.e. figure), foggy"* and so forth.
- Allows a user to add words to the lexicon's (the dictionary used for look-up) content, usually referred to as the "*user dictionary*" [19].
- Provide synonyms and antonyms for a given word.
- Advanced software also includes the ability to supply a word's equivalent in another language, for example the Afrikaans equivalent for the English word "*knife*" is "*mes*", the South Sotho word "*dijo*" refers to the English word "*food*" and so forth.
- Another advanced feature of spell checkers and correctors like the above mentioned, when implemented in a word processor, is to offer the ability to recognise homophones, i.e. words that sound the same but are spelled differently.

The system incorporates algorithm-based text error detection whilst employing multi-threading technology to facilitate better algorithm performance. If a spelling error is detected, the word is flagged and brought under the user's attention. Suggestions are made with regard to the correct spelling of the misspelled words and listed accordingly. In cases where there is no a large deviation between the incorrectly spelled word and the correct example, according to the dictionary, the mistake will be automatically corrected. This process is facilitated by modules that contain code for the various processes that need to be executed, passing data to-and-fro the database and the main program.

The following chapter focuses on the field survey done (in the form of a questionnaire), mentioned in chapter 1, as well as a sub-set of statistics which was extracted from the survey.

Chapter 3

3. SPELLING SKILLS

This chapter reviews a questionnaire that a focus group was subjected to during the course of October 2007. We outline the reasons for this part of the study as well as an explanation of the questionnaire itself and the statistics extracted from it [Appendix C]**.**

3.1 The need for the questionnaire

There are no official statistics regarding the number of spelling mistakes made by the average student or scholar nor is there any other data available concerning this matter. For example, we would have liked to know how spelling has deteriorated or improved over specific time frames. Hence, the researcher decided to embark on this experiment in the form of an opinion poll based on a questionnaire. The researcher wanted to reach some reasonably accurate conclusions to the following question: "*What is the state of our spelling skills these days, especially when focussing on the youth?*"

The opinion poll was targeted to a specific focus group; second year students, twenty years of age studying at the Central University of Technology, Free State. The rationale behind this decision was that these students have been out of school for approximately two years, with at least a year-and-a-half post secondary education prior to completing the questionnaire, which would enable them to apply the spelling skills obtained during their school career. This meant that their completion of the questionnaire could indicate either an improvement or deterioration of their spelling skills.

Acquiring statistics vis-à-vis how spelling has deteriorated, or improved, would mean that a statistician would have had to study spelling tendencies from, for example, the early nineteen-ninety's until the present. Needless to say, this

was not done, mainly because in the past there was very little statistical focus on spelling skills.

The purpose of the questionnaire was four fold. Firstly, to acquire a general view of what the state of spelling skills is within the focus group and to identify factors that influence how the subjects spelled in general. Secondly, for the (recent) data, relevant to this focus group, to be extracted and reworked into usable information through statistical analysis. Thirdly, to discover what extra features users would like to have in a spell checker and corrector and finally to ascertain if the development of a spelling checker and corrector for South Sotho can be justified.

3.2 The questionnaire and choice of language

During the course of October 2007, a focus group of forty second year students of the Central University of Technology, Freestate, was subjected to a questionnaire [Appendix C]. The subjects participating in this experiment were not asked to provide their names, thus making the results obtained more sincere as well as more reliable. The questionnaire was compiled using the design principles and techniques discussed in "*Human Computer Interaction, Serengul Smith-Atakan, 2006"* [26]. A copy of the questionnaire is shown in Appendix C.

Participants were provided with a series of questions ranging from open-ended questions, to allow elaboration on the individual's views, allowing us to identify certain trends; questions in which a simple "*Yes*" or "*No*" answer would suffice as well as questions where the individuals had to rate certain criteria according to a fixed supplied scale ranging from 1 to 5.

The initial focus group that responded to a call for volunteers consisted of a multi-lingual combination of students including Afrikaans, Sotho, Tswana, Xhosa and Zulu speaking individuals. Due to the multi-lingual nature of the focus group, stated above, the format of the questionnaire posed some

potential problems. If the questionnaire was based only on each language identified, the focus group might have become too small, which could then have affected the accuracy of the statistical analysis. Furthermore, the group did not consist of equal proportions of a certain language spoken; for example, the group did not consist of ten Afrikaans speaking individuals, ten Sotho speaking individuals, ten Xhosa speaking individuals, and so on; and this posed an additional problem.

Some solutions to this problem were:

1. To proportion the group equally according to languages spoken and compile a questionnaire in a specific language for a specific sub-group.
2. To find a common language in which to compile the questionnaire.
3. To compile a questionnaire for a group of South Sotho speaking individuals only.

The first solution posed significant problems that could influence the experiment negatively, the largest being that the researcher would have had to employ a specialist for compiling and validating the questionnaire in each language. This would have been time consuming, especially if the results had to be obtained within a specific time frame. Thus, the first option would have been a tedious, expensive and time consuming task involving the creation of a subset of five questionnaires, in different languages, containing the same information and keeping the standard of the questionnaire the same, not to mention extracting the results from the questionnaire. One of the reasons the researcher mentions the latter, is that intentional spelling mistakes were made in the questionnaire, to see if the participants would be able to identify them. The historical background of a specific student might also have influenced the outcome of the questionnaire's statistical analysis if this option was utilized.

In the case of the second option, extracting results reliably would have been a drawback, because the results for each sub-group might differ, e.g. the Zulu speaking sub-group might yield better results than the Tswana speaking sub-group. Also, if five groups completed the questionnaire separately, in their

own home-tongue, five sets of results would have had to be combined, which was not sought after and would have yielded inaccurate results.

The third option provided the best trade-off. It was then decided to assemble a new group of forty South Sotho speaking students to complete a South Sotho-based questionnaire, which was compiled with the assistance of a South Sotho speaking lecturer at the Central University of Technology, in conjunction with the researcher. The aim was ultimately to develop a spell checker and corrector application system for South Sotho.

3.3 The statistics

The participants were asked to supply their matric results for South Sotho, anonymously. Thirty three of the forty subjects had a 7% difference or less in their scores. That meant that 82.5% of the focus group was more than proficient to understand and answer the questionnaire in their home tongue. Five of the participants had a marked difference of 8% to 10% between their mark obtained when compared to the former and two in the group had a marked difference of 11% or more when compared to the pre-mentioned part of the group. This was deemed acceptable.

In the sub-sections to follow, the data extracted was reworked into statistical values. Each sub-section explains the statistical values under consideration.

3.3.1 Deterioration of spelling skills among South Africans

Question one in the study asked the subjects if they were of the opinion that South Africans' ability to spell correctly was deteriorating or not. Figure 3.1 depicts the results. As Figure 3.1 illustrates, 85% of the focus group felt that spelling skills among all South Africans have deteriorated in general. This means that only six out of the group of forty were of the opposite opinion.

Opinion among focus group whether spelling skills have deteriorated

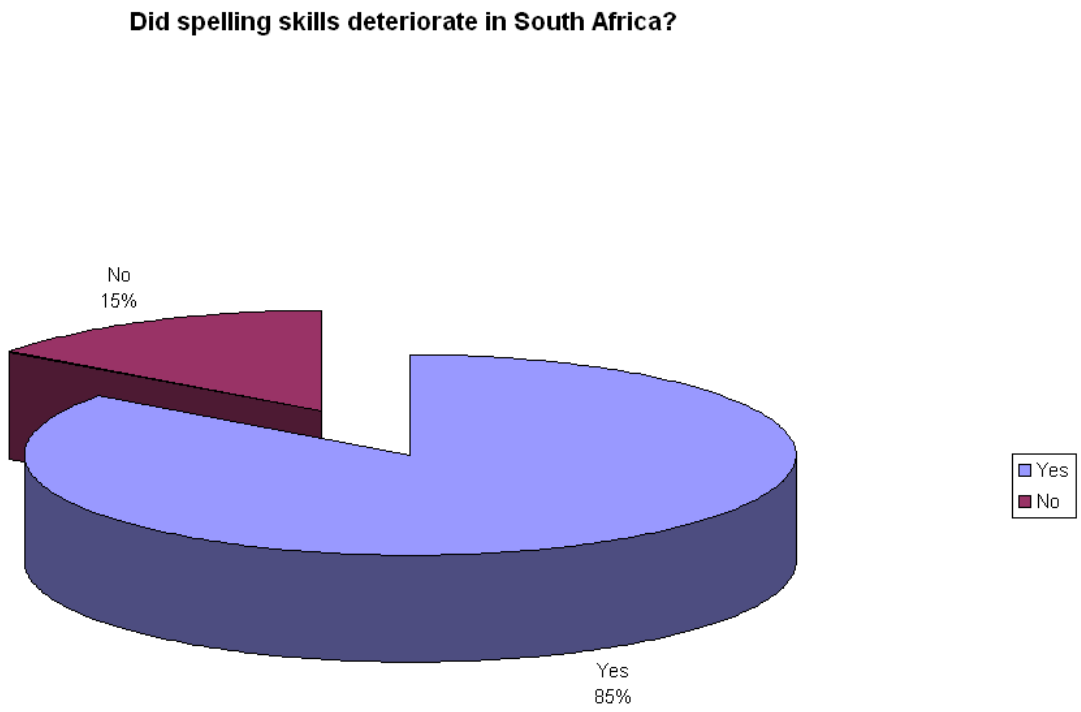


Figure 3.1: Deterioration of spelling skills among South Africans

3.3.2 Deterioration of spelling skills among South Sotho speakers

Although all of the participants were South Sotho speaking, the members of the focus group were instructed to apply this question to all languages they write in. Figure 3.2 shows the outcome. Seventy five percent (30 out of the 40 participants) were of the opinion that the spelling skills of speakers of indigenous South African languages were deteriorating and 25% thought otherwise.

Deterioration of spelling skills among South Sotho speaking individuals: Yes or No?

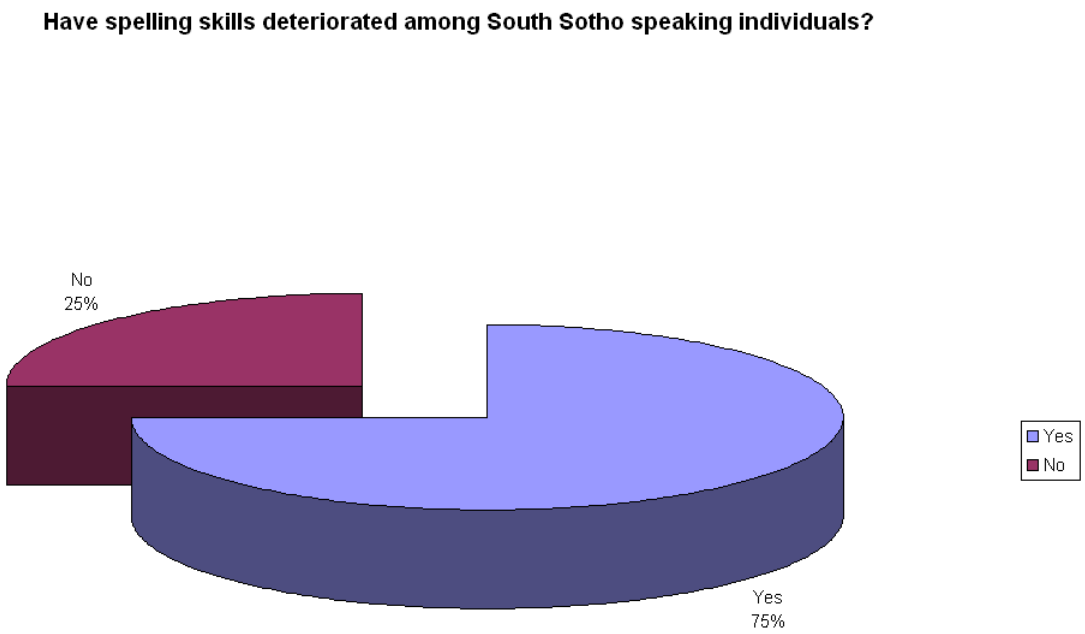


Figure 3.2: Deterioration in spelling skills among South Sotho speakers

3.3.3 Contributing factors to the deterioration of spelling skills.

In question three the members of the focus group were asked to list factors like, for example, technology that they felt contributed to the downward spiral of spelling skills. The results are illustrated in Figure 3.3:

A total of eleven factors were mentioned. These included: the fact that some people came from *disadvantaged communities*, the use of *cell phones* for communication (especially text messaging), the *music* listened to by the youth, *television programs* watched and/or *radio programmes* listened to as well as the media *presenters* regarding the language they use. Other factors included the use of *E-Mail*, the so-called "*slang"* terms used in ordinary conversation, *magazines* read, especially youth oriented magazines, *spell checkers* in use, the *mixing of spoken languages* as well as the fact that people *read less* regarding wholesome material consisting of good language use, grammar, and so on.

Poll opinion: What factors contribute to the deterioration of spelling skills?

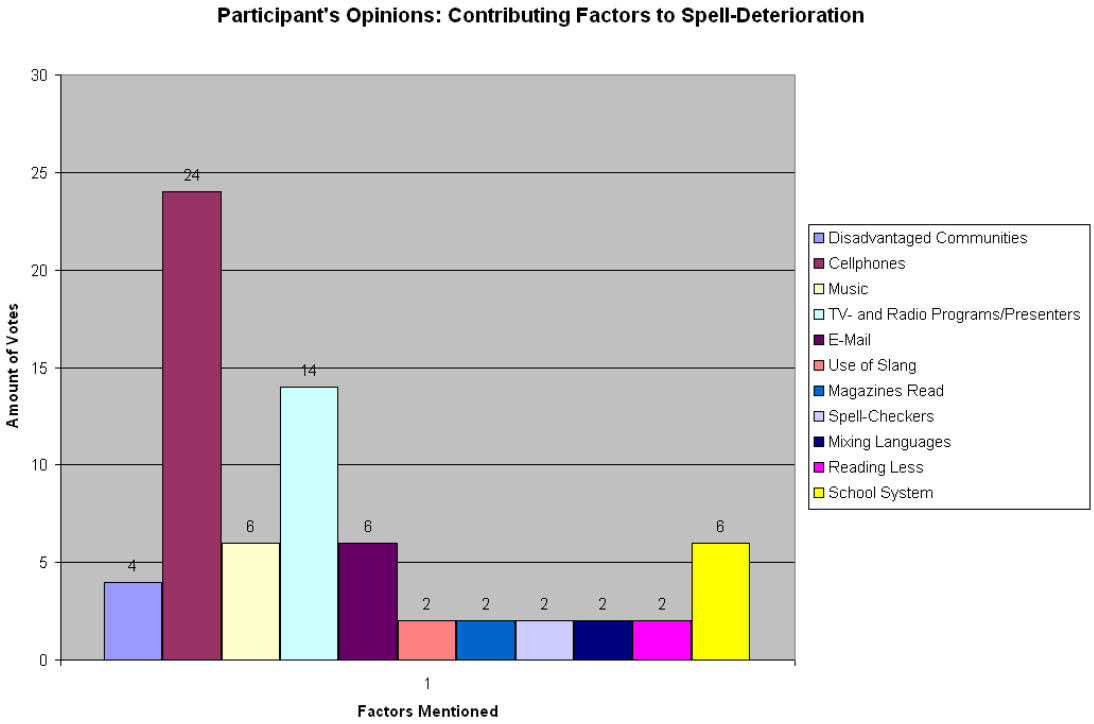


Figure 3.3: Contributing factors to the deterioration of spelling skills

Additionally, the general belief is that the current *school system* does not focus enough on the development of spelling skills, not only by the focus group, but also according to the contributions on the blog: *Grammar-deteriorating-teaching-profession* [18]. The top scoring culprits were identified by the focus group as the use of cellular phones, scoring twenty-four votes out of a total of seventy, which is 34% of the total. The television- and radio programmes in circulation as well as the language used by the presenters, scored fourteen out of seventy, meaning they carry 20% of the total blame, as per opinion of the focus group.

The fact that television emerged as the second most important contributor to the deteriorating spelling skills came as no surprise. Presenters and other media personalities use words such as "*fschizzel*", which has no apparent meaning, spelling or entry in the Oxford Thesaurus. Our own national broadcaster, the South African Broadcasting Corporation (SABC), also

contributed to this rating by the focus group. For example, SABC 1 uses the term like "*fo sho*", written on the screen instead of the correct term "*for sure*". A person cognitively uses how the word sounds and relates it to the spelling of words that he/she couples with these slang terms.

The opinion that *E-Mail* contributes to spell deterioration is quite peculiar, considering that writing E-Mail is governed by the same rules as writing a letter. It has long been suggested by scholars (i.e. the academic community) [34] that reading novels, wholesome literature and so forth, leads to spelling proficiency. It is the researcher's opinion that *reading less* had to score much higher as a contributing factor to the deterioration of spelling skills. Materials such as teen magazines and so forth also use "slang" in their articles, which could promote the deterioration of spelling skills. Because there are no entries for some of these words in dictionaries (yet), the journalist or columnist for the magazine has free reign to spell these words as he or she sees fit, which may also differ from journalist (columnist) to journalist (columnist).

Spell checkers scored 5% as a contributing factor to spelling skills deteriorating. It cannot be ignored that this might very well be a contributing factor since the user of such a programme can promote a trend of idleness in that the user knows that the chances are very good that the correct spelling of the word will be suggested by the application. This trade-off is acceptable when one regards this type of software as a tool to produce error free documents for industry, for example, legal documents and so on; as humans are prone to making errors, albeit spelling mistakes. It should not be regarded as a tool to justify laziness where a student or scholar could care less about the correct spelling of a word, because he/she knows that the software will help rectify the mistake. Nor should it be used as a learning tool or aid to one's own poor spelling-ability at best.

3.3.4 Effect of education system in improving spelling skills

The respondents were also asked the following question: *Could our education system do more to improve spelling skills?* In response to this question, 65%

of the focus group were of the opinion (having left school only 2 years prior to the participation in this study) that the current education system was lacking in producing mechanisms to help improve spelling skills of students. The statistics are illustrated in Figure 3.4.

Suggestions of where the system could improve ranged from subtracting marks for every 5 spelling mistakes made in written work to the writing of more essays. Some were of the opinion that it was the student's responsibility to keep informed regarding how to spell, whist other opinions suggested spelling bee's to improve spelling skills from a young age. One response was quite shocking as it stated "*my school didn't pay any attention to spelling and didn't subtract marks for incorrect spelling*".

Should schools do more to promote sound spelling skills?

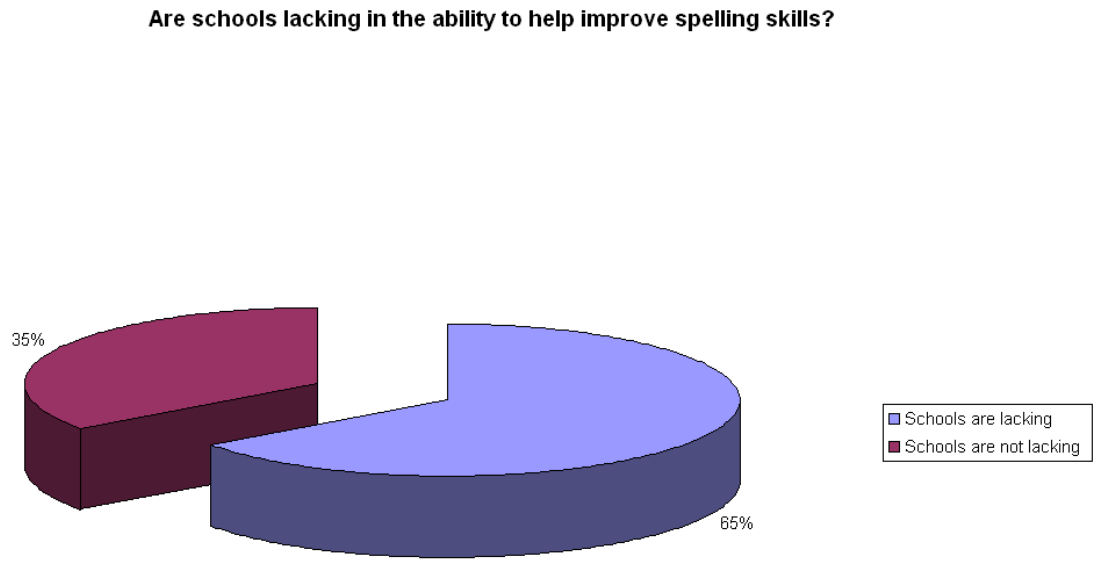


Figure 3.4: Effect of the education system in improving spelling skills

3.3.5 Mixing of languages (Code mixing)

The fifth question was regarding languages: *Did the participants feel that mixing different languages contributed to the problem being investigated?* When speaking of the mixing of languages, we do not refer to one's bilingual ability or proficiency in more than two languages; reference is made to polluting a certain language with phrases and words from other languages.

Mixing of Afrikaans, English, Dutch, Zulu, Xhosa and related languages has been happening in South Africa for decades, and this lingua franca is called *Fanagolo*. South Africa's indigenous languages are also being mixed by the youth today. The problem is that mixed languages are not based on any vernacular or clear set of rules, and as such spelling becomes a problem when meanings of words differ largely because spelling is now based on what the users perceive it to be.

Thus, does mixing our language have an effect on spelling? Only 5% were of the opinion that when one mixes one's language one's ability to spell correctly will not be affected adversely. Figure 3.5 shows what the respondents thought:

Mixing languages: Does it affect spelling skills?

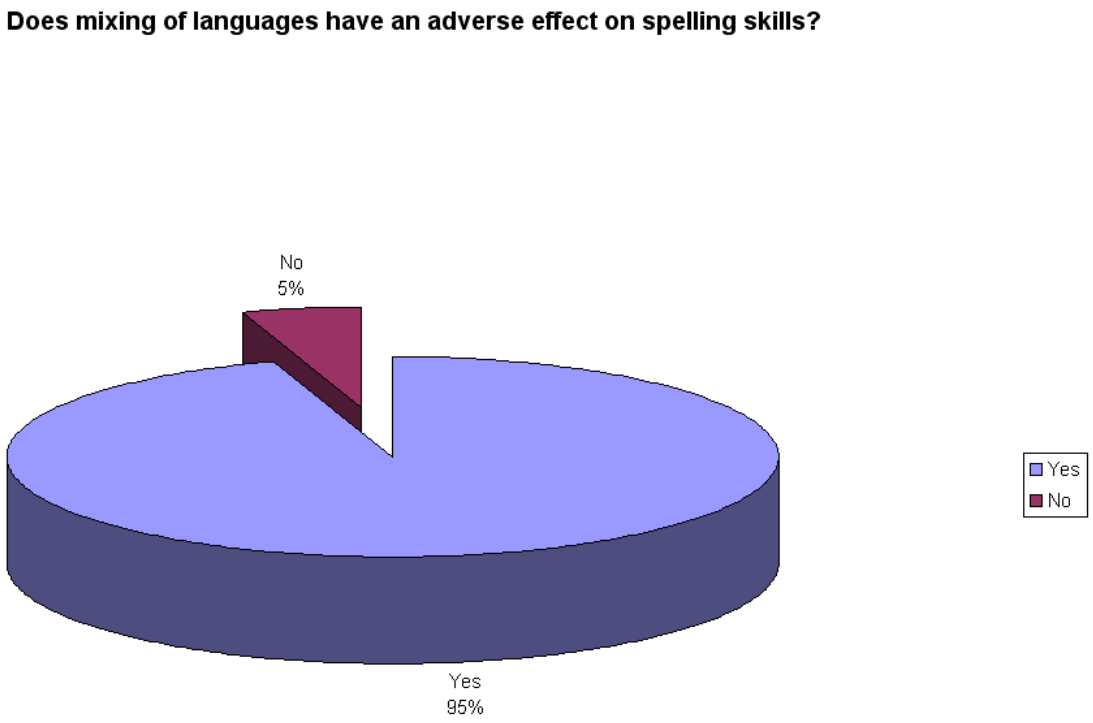


Figure 3.5: The effect of the mixing languages on spelling skills

3.3.6 The effect of reading printed material on spelling skills

The researcher also posed the following question: *Will reading novels, scientific papers, and so on, improve spelling skills or not?* As stated earlier,

there are arguments that suggest that reading such material should improve spelling skills. Only 3%, meaning only one person in the focus group, was of the opinion that reading does not help to improve spelling skills. The catch phrase here is reading "wholesome material", like a Stephen King novel, the daily newspaper, scientific magazines, and so on. Figure 3.6 conveys their views.

Does reading help improve the spelling skill-set?

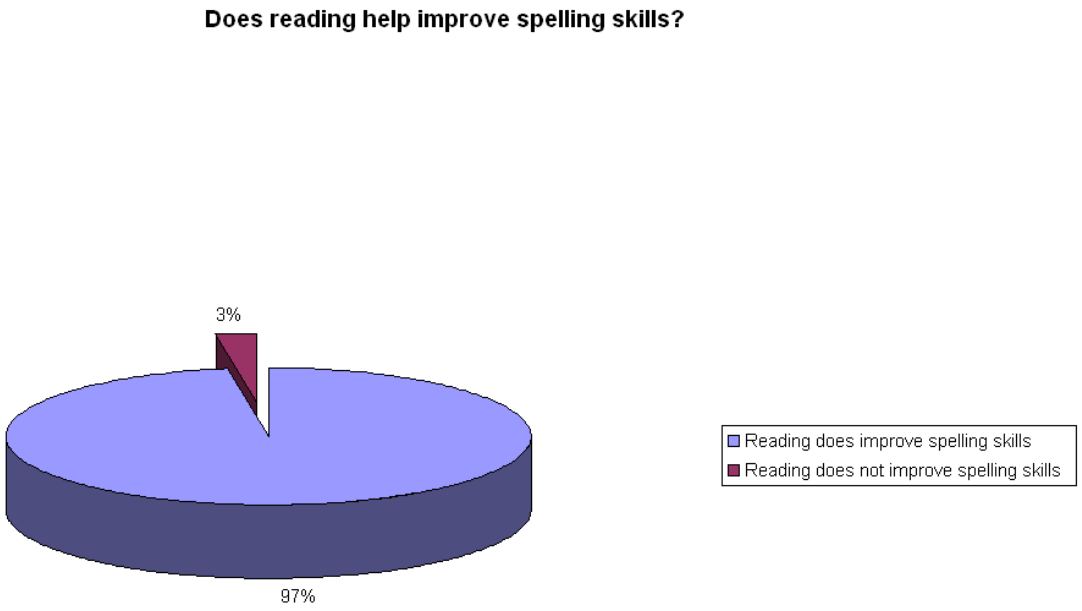


Figure 3.6: Does reading help to improve spelling skills?

3.3.7 Spell checker and corrector influences

This question elaborated on question three of the questionnaire, as it is relevant to the research conducted. Question nine of the study asked the focus group how much they used software such as MS Word as well as what they thought the positive and negative aspects of such a programme were. In a study like this, one cannot ignore the fact that software like MS Word could contribute to the declining curve of spelling skills within a socio-economic environment. The trade-offs should be carefully studied and its implementation and use (or not) must then be based on those findings. Maybe scholars and students should not be encouraged to use these

programmes at school level for assignments like essays and so forth. Rather, maybe the use of a good, old-fashioned, dictionary should be encouraged and promoted.

Figures 3.7, 3.8 and 3.9 illustrate the views of the focus group with respect to the frequency of using a spell checker, its positive attributes and its negative aspects respectively.

Are spell checkers used more often than not?

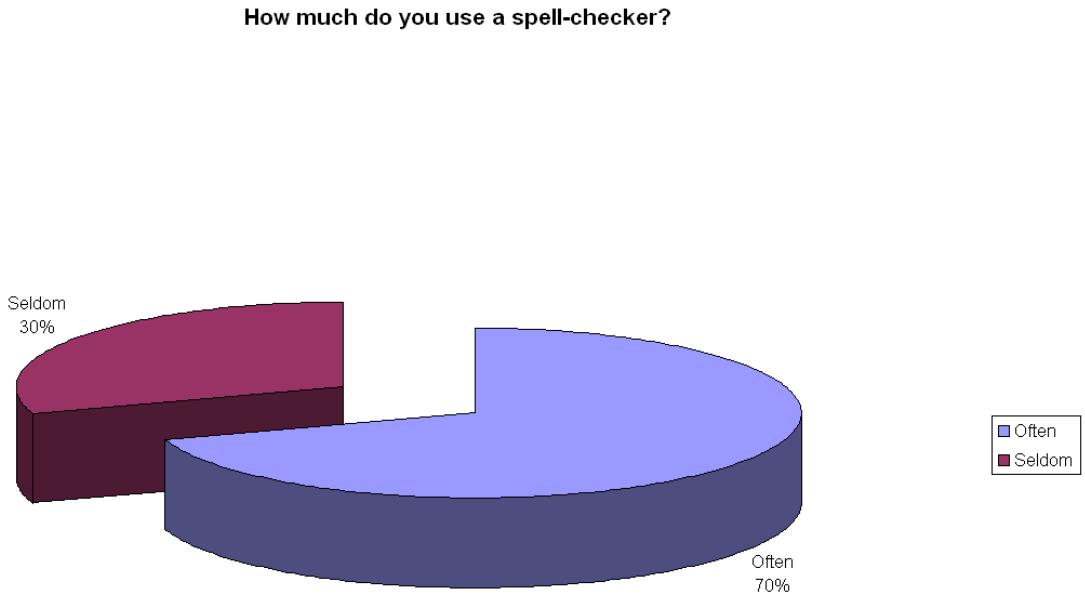


Figure 3.7: How often is a spell checker used?

Twenty eight of the forty participants responded that they used some kind of spell checker quite often, whereas 30% said that they did not. It can be suggested that the latter could be due to the fact that members of the focus group did not have access to computer resources on a daily basis or that the spell checkers commonly available do not support the language the user wants to use. It should be mentioned that the group consisted of students in their second year of study, and their “lab-times”, where they have constant access to computers, exceeded five hours per week.

The next figure highlights the positive attributes of spell checkers listed by the focus group. As shown in Figure 3.8, the participants suggested three

positive attributes of spell checkers. These included the *suggestion of alternative words* by the software, the view that the users *save time* in task completion when using the software as well as the most obvious positive attribute, *assisting in fewer spelling mistakes made in important documents.* These attributes scored 20%, 40% and 40% of the votes respectively.

Poll opinion: Positive attributes of spell checkers

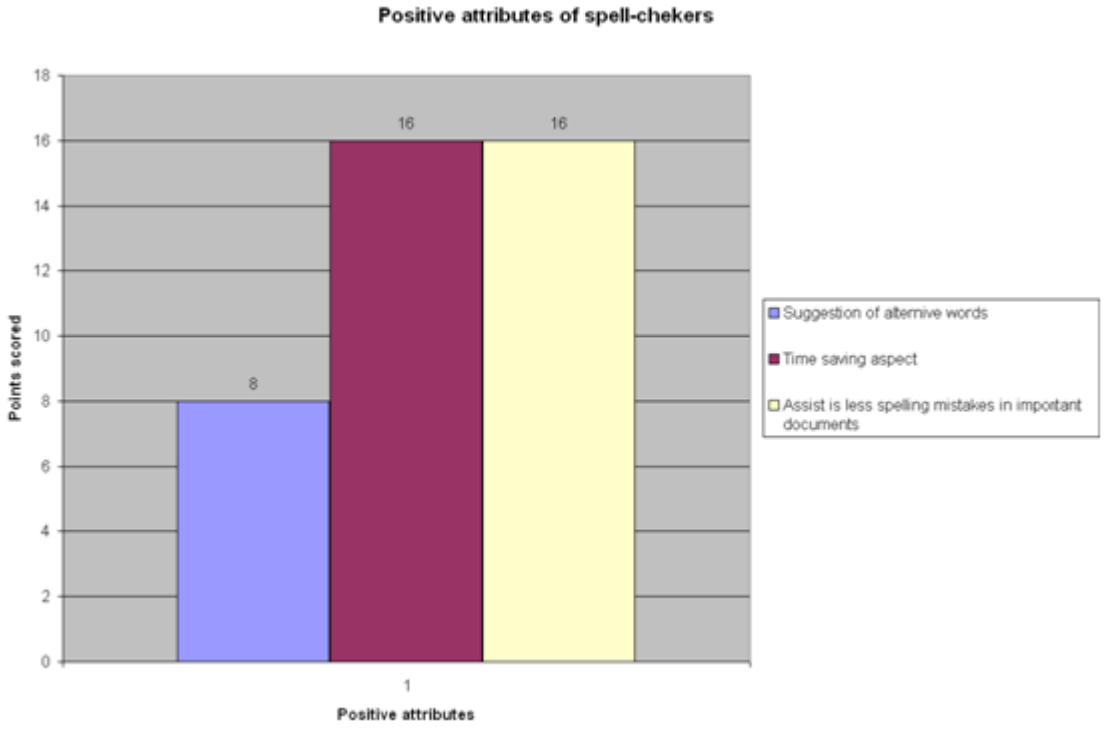


Figure 3.8: Positive attributes of spell checkers

Poll opinion: Negative attributes of spell checkers

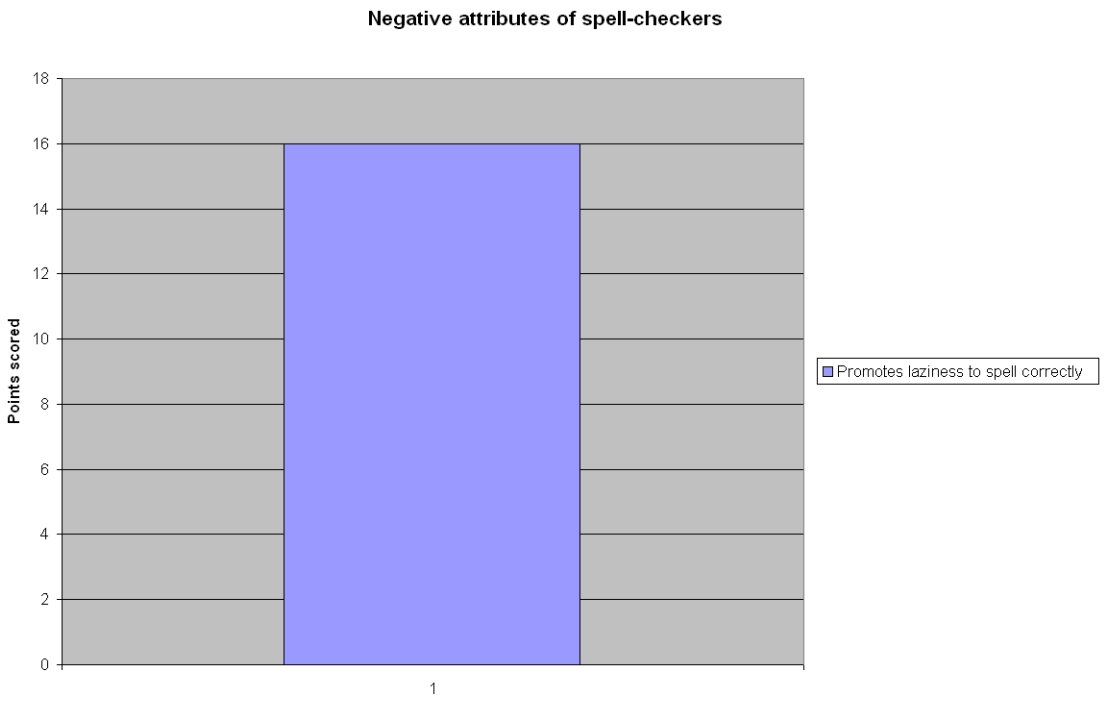


Figure 3.9: Negative attributes of spell checkers

Figure 3.9 suggests that 40% of the focus group were of the opinion that using spell checkers creates a culture of laziness among its users, i.e. not caring how they spelt words since they knew that the software would alert them if they made any spelling mistakes. This was the only negative attribute listed by the whole of the focus group, thus, the conclusion could be made that 100% of the participants that listed negative attributes regarding spell checkers, felt that it promoted laziness in spelling correctly.

3.3.8 Additional desirable features for spell checkers

The next question raised wanted the participants' view on extra features they wanted to be able to use in spell checking software. Out of the twenty suggestions received (i.e. only twenty participants of the focus group listed extra features), 30% listed that *more names and* surnames should be recognized by the software. In my view, this is a Utopian idea, because many names are combinations of other names. A list of the most popular names and surnames already form part of, for example, MS Word. Another 60%

listed that they would like to see *the explanation of words and phrases, like as in the dictionary,* to be included. Finally, 10% wanted to see *more languages, including their native language*, included in such software. Figure 3.10 illustrates these wishes expressed by the focus group.

Opinion poll: What extra features should be included in commonly available spell checkers?

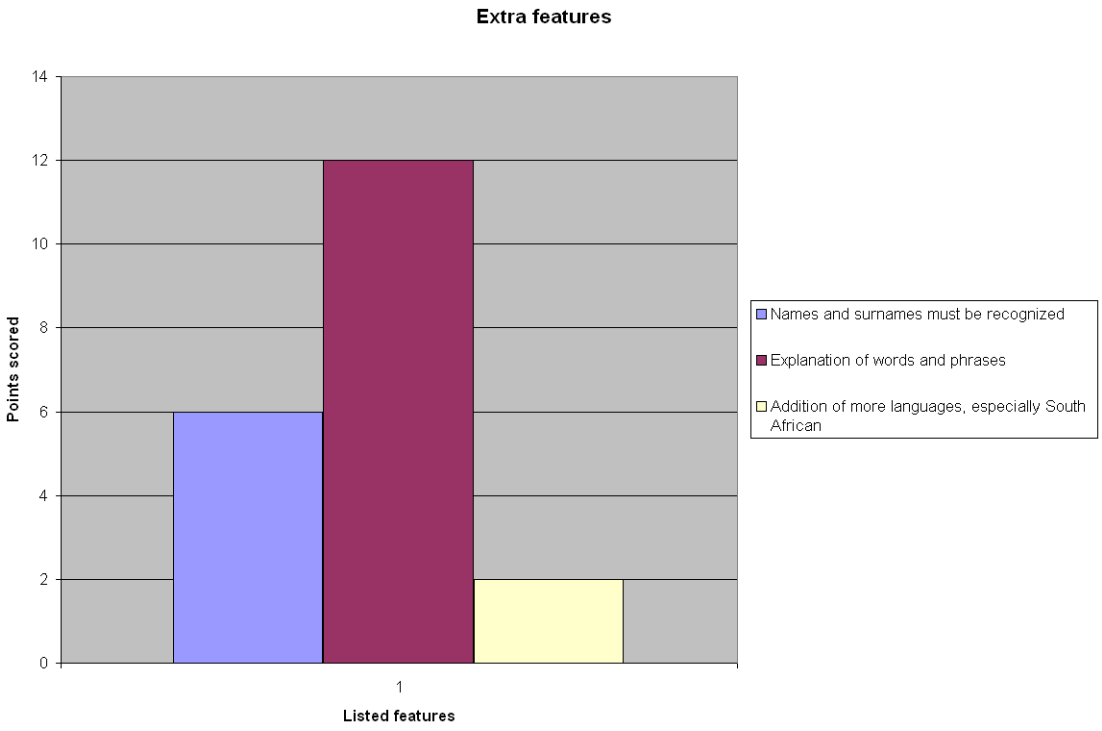


Figure 3.10: Extra features to include in a spell checker

3.3.9 Scale-based rating of the contributing factors

Question 12 supplied the focus group with a series of factors that could be perceived as being contributing factors of spelling skill deterioration in society, both applicable to South Africa as well as globally.

Participants had to rate factors by awarding points, from a provided scale ranging from one to five. A value of one would represent the lowest, i.e. not contributing to spell skills deterioration at all, and five representing the

highest, i.e. this factor has a big influence on the way spelling skills deteriorate. A summary of the scale values is given below.

1. Not a contributing factor at all, i.e. this factor has an accountability of 0%.
2. Cannot be held totally or directly accountable.
3. Has accountability of approximately 50%.
4. Seen as a factor that can be viewed as contributing extensively regarding the way spelling skills deteriorate.
5. Can be held very accountable and has a contribution factor of 90% or more on how spelling skills deteriorate.

The following series of figures (Figure 3.11 to Figure 3.19) depict the frequency of each of the above mentioned range-ratings, specific to only one factor, for example, *Television*.

Frequency of range values: Mixing languages

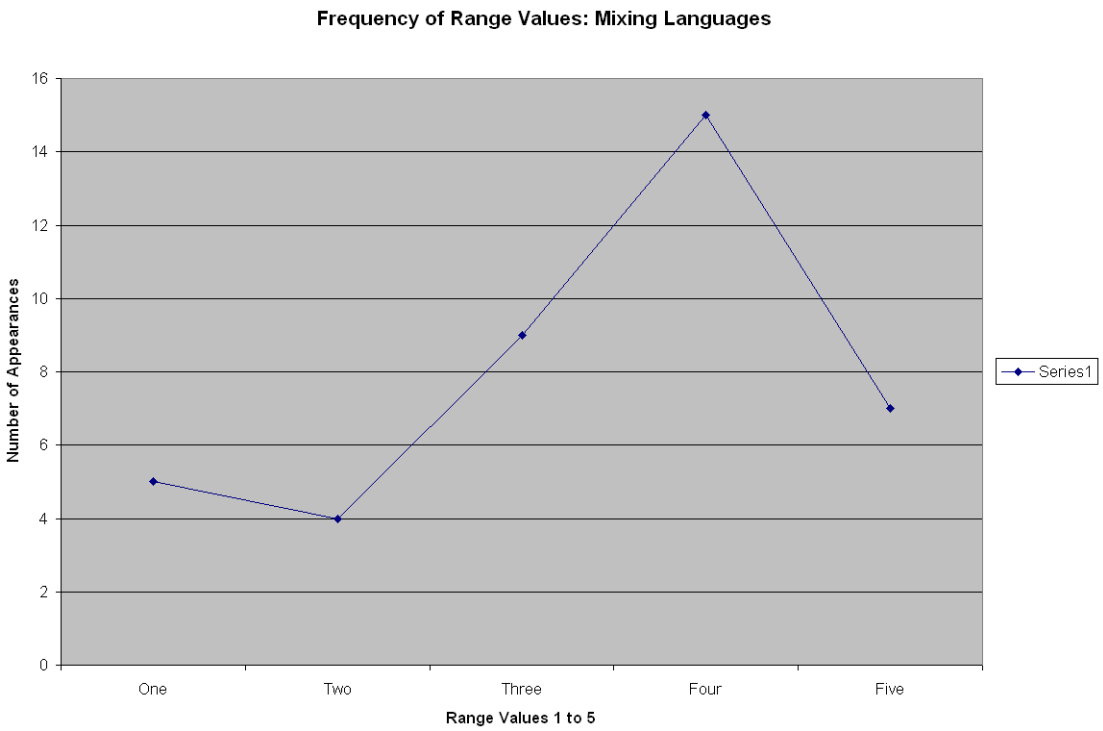


Figure 3.11: Mixing languages

Figure 3.11 depicts the scale-based rating of the effect of *Mixing of Languages* as contributing factor to spelling skills deterioration. The chart shows that out of the forty participants, fifteen assigned a value of four to *mixing languages* as a factor contributing to spelling skills that are deteriorating. This relates to 37.5% of the total vote. The obvious spike in the chart is a clear indicator that the *mixing of languages* is seen as an extensive contributing factor with regard to the way spelling skills deteriorates.

Figure 3.12 communicates the views of the focus group with regard to the *music listened to by people, especially music resembling the kwaito/rap genre.* The fact that range value five is elevated above its counterparts indicates that this offender "*can be held very accountable and has a contribution factor of 90% or more on how spelling skills deteriorate*" strongly suggest that this factor can be viewed as a major contributor to the deterioration of spelling skills already mentioned. Kwaito is well known for its language mixing whereas rap is just as notorious for its use of English "slang" and the "adjustment" of phrases and words, for example, using the term "*I aksed you a question*" in stead of "*I asked you a question*".

As can be seen from Figure 3.12, seventeen out of the forty respondents rated this factor with a five, relating to 42.5% of the total number of participants. An additional 35% rated this factor with a value of four.

Frequency of range values: Music listened to

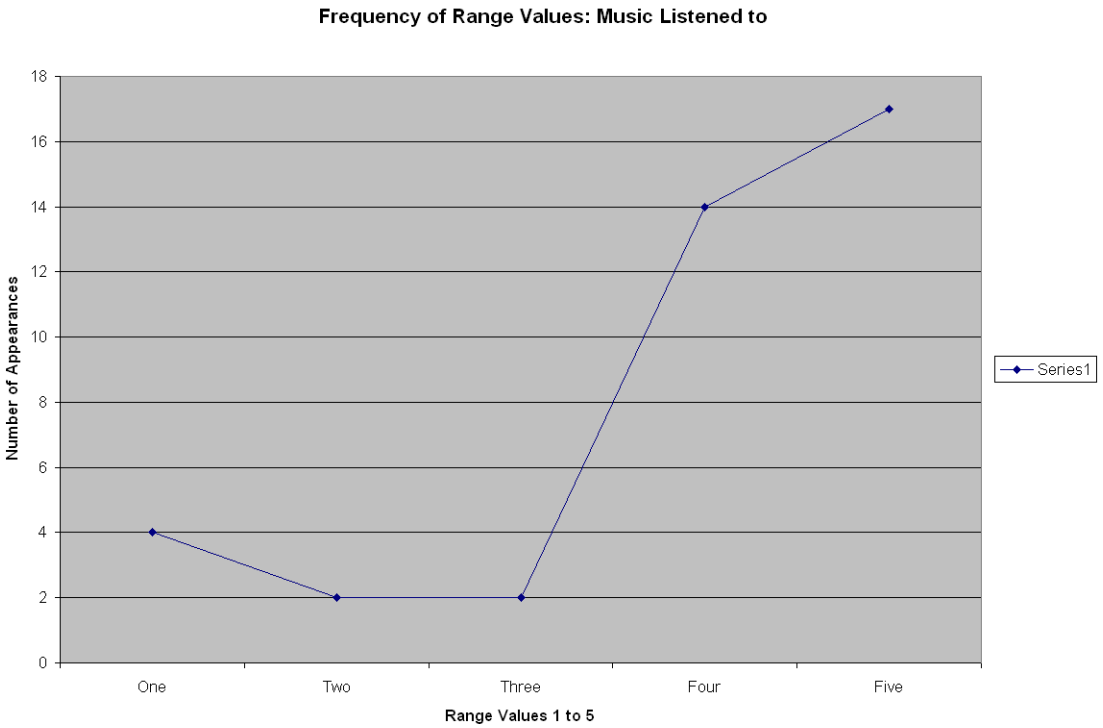


Figure 3.12: Music listened to

Figure 3.13 depicts the *performance of schools* vis-à-vis taking responsibility in addressing the deterioration of spelling skills.

Frequency of range values: Accountability of the school system

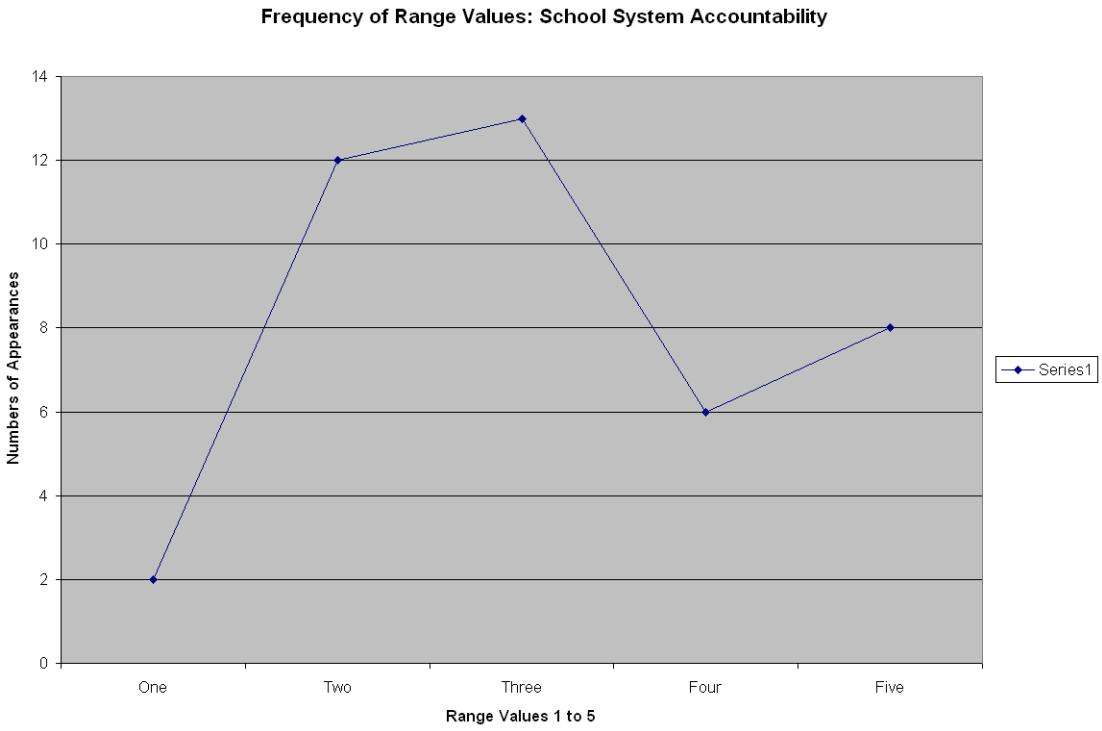


Figure 3.13: Accountability of school system

In Figure 3.13, the scale value three has a definite spike conveying the message that the current school system is indeed lacking based on the views of the focus group. According to scale value three's description, 32.5% of the group felt this factor carried a contribution weight of 50% concerning accountability for the problem.

Figure 3.14 expresses the views of the participants with regard to *radio programmes on the radio and radio "lingo" used by presenters* as a factor contributing to the deteriorating spelling skills. The opinion is clear: according to this focus group, radio lingo and the programmes aired on radio do not have a direct effect on the way people spell.

Frequency of range values: Use of radio "lingo" and radio stations listened to

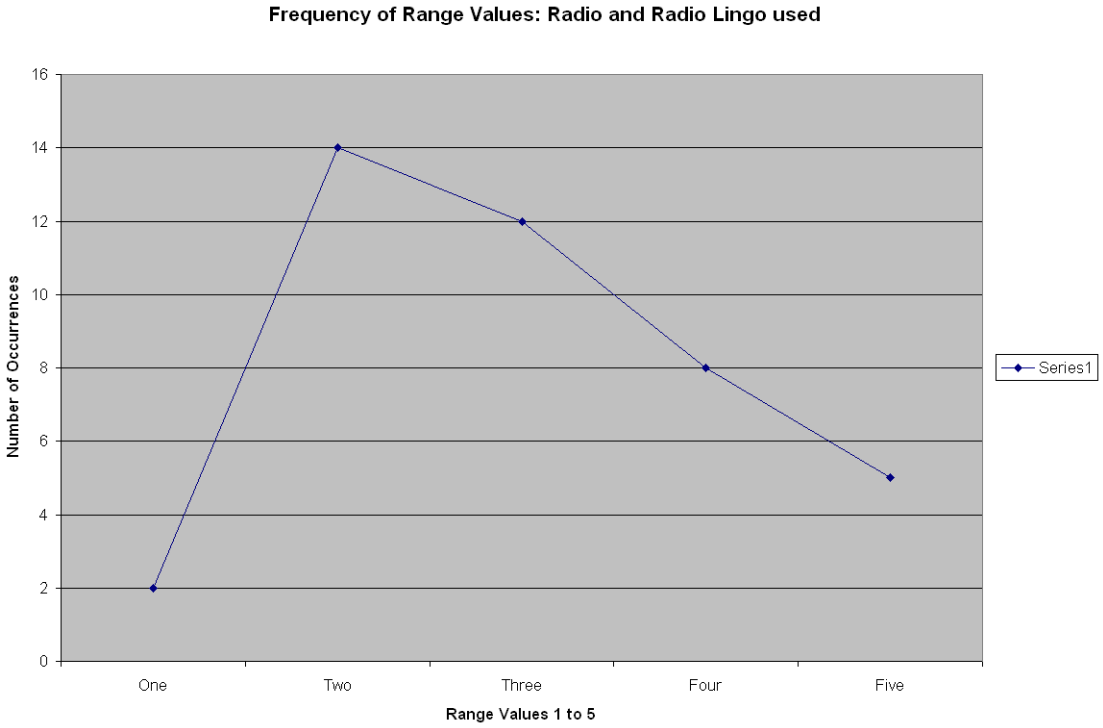


Figure 3.14: Use of radio lingo and radio stations listened to

Some scholars and researchers [34] have suggested that another, maybe not so obvious factor, that actually improves the spelling skills of an individual, is *reading*. Not just any reading, but, print material such as papers, novels, science papers, etc. According to the opinion poll, young people spent less time reading these types of printed material and Figure 3.15 clearly shows the view of the participants in this regard. The additional question the researcher asked to himself at this stage was: *Will a person take on the responsibility of picking up a book and starting to read healthy and/or informative material, or not?*

As can be seen from Figure 3.15, 40% (sixteen out of forty) of the group was of the opinion that *reading less* has an adverse effect on the spelling skill-set of an individual. An additional 22.5% gave this factor a rating of four, which means that *reading less* can be viewed as contributing extensively to the way spelling skills deteriorate.

Frequency of range values: Contribution of reading less on spelling-deterioration

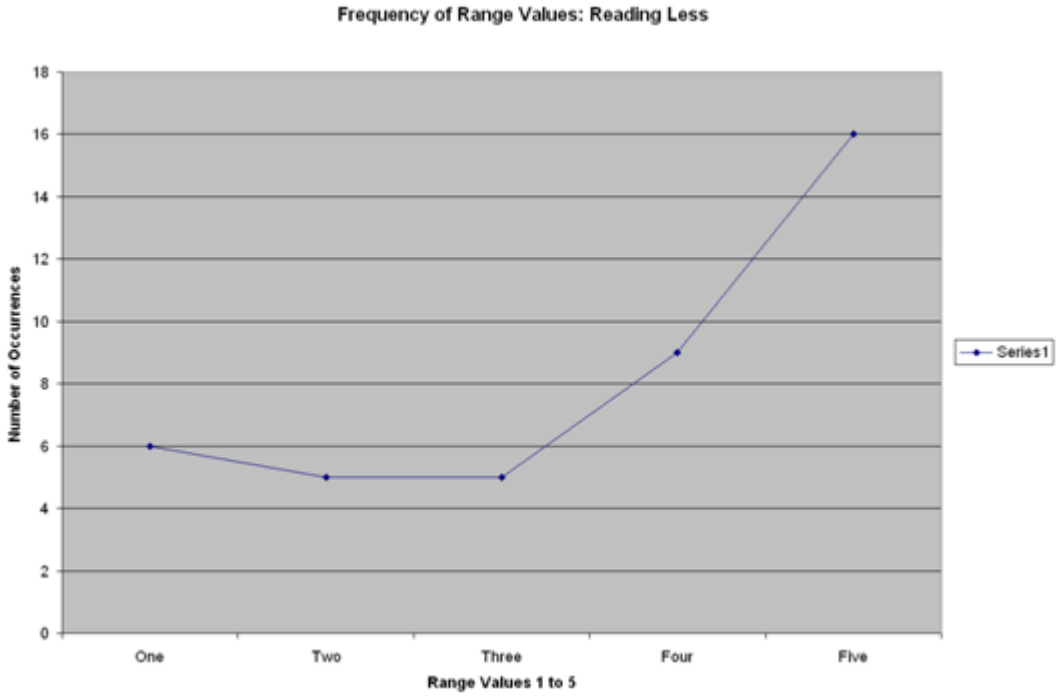


Figure 3.15: Reading less

Reading youth magazines was not put under further statistical scrutiny in this study, due to the facts shown in Figure 3.3 under '*Magazines Read*'. It is accepted and expected that the use of good spelling and grammar is checked and promoted by magazine editors.

The next contributing factor under investigation was *television*. How do the programs we watch, the presenters' dialogue, advertisements, broadcasts using television as a medium and so forth, contribute to the way we spell? As can be seen from Figure 3.16, 25% responded with a rating of three, indicating that *television* could be held somewhat responsible for the decline in spelling skills. If this result is combined with the 22.5% and the 20% who gave this factor a rating of five and four respectively, the assumption can be made that television does play a more than average role as an offender when it comes to a contributing factor to spell deterioration.

Frequency of range values: Television

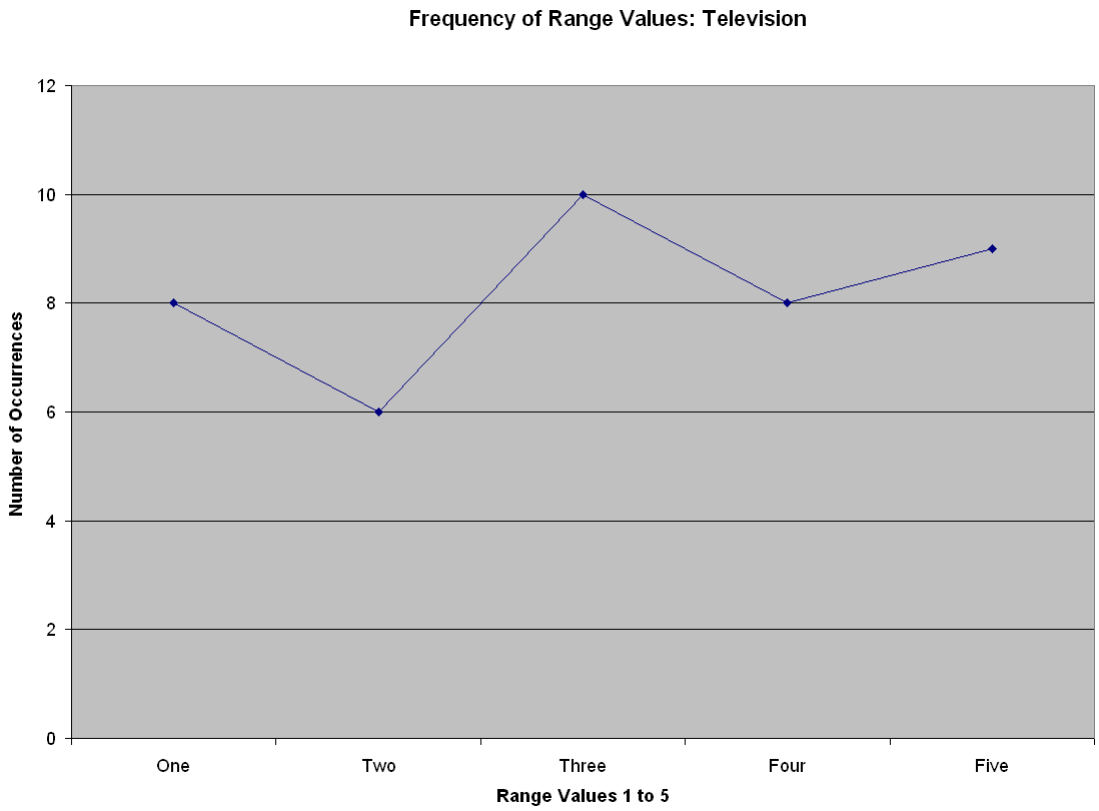


Figure 3.16: Television

The information captured in the following figure might be regarded as the most significant and informative of this chapter. Allegations made by the public, by professionals, for example, teachers as well as in the print media [8] regarding how spelling has deteriorated due to the fact that more and more people are using *cellular phones* for *text messaging* are becoming more and more. Using the cellular phones as a medium to communicate has become very popular. Teachers are reporting spelling chaos in schools due to the extensive use of cellular phone text messaging by students. Apparently students are using so-called *sms language* (e.g. instead of asking "*How are you?*", in *sms* dialect one would ask "*How r u?*") in essays, to answer question papers and so on. Figure 3.17 depicts the scale-based rating of the contribution of text messaging to the deterioration of spelling skills. Interestingly enough, the two most dominant views are on the opposite sides of the range when it came to this factor.

Six respondents' (15%) opinions were that text messaging using a cellular phone does not contribute to the downward spiral of spelling skills, while an extraordinary 77.5% supported popular belief that this factor is a major contributor of spelling skills deterioration. The views are very clear: text messaging does contribute to the posed problem. Note that no participant rated this factor with either a scale rating of two or four, conveying their views very firmly.

Frequency of range values: Text messaging

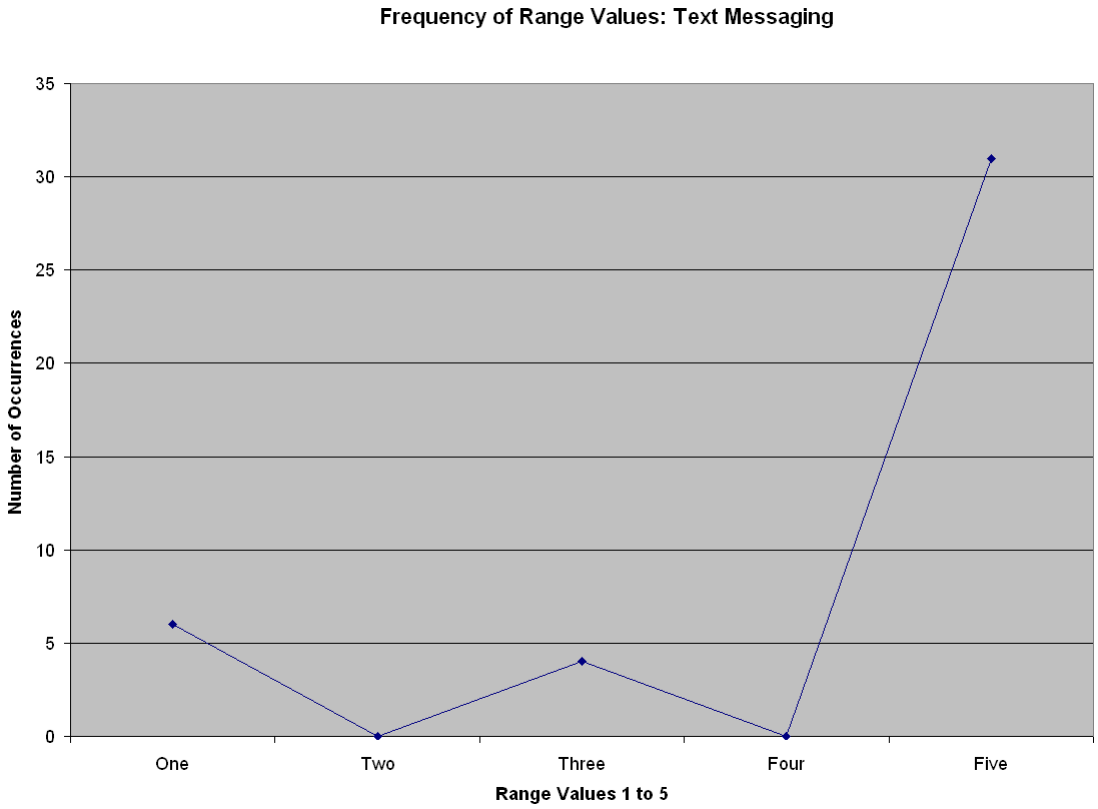


Figure 3.17: Text messaging

The last factor to be considered was the *spell checkers*. Is the trade-off between the production of error-free documents and spelling skill deterioration worth it? It can certainly not be expected of industry to leave the production of important, error free, documents to the mercy of the spelling skill-set of most of today's "spellers"? Where should the line be drawn? Again the word "*responsibility*" comes to mind. Each person must take responsibility to

develop and improve their own spelling skills-set. Is spell checking software a large contributing factor to deteriorating spelling skills?

Figure 3.18 shows the opinions of the members of the focus group. The chart shows that 32.5% of the group did not consider spell checking software as a contributor to the declining spelling skills. The education system came under scrutiny when the researcher considered this factor, and a few questions could be posed: Should a scholar not be able to spell relatively well upon leaving school? Should hand-written assignments not be promoted as opposed to word processor corrected printed versions?

Frequency of range values: Spell checkers as a contributing factor to spell deterioration

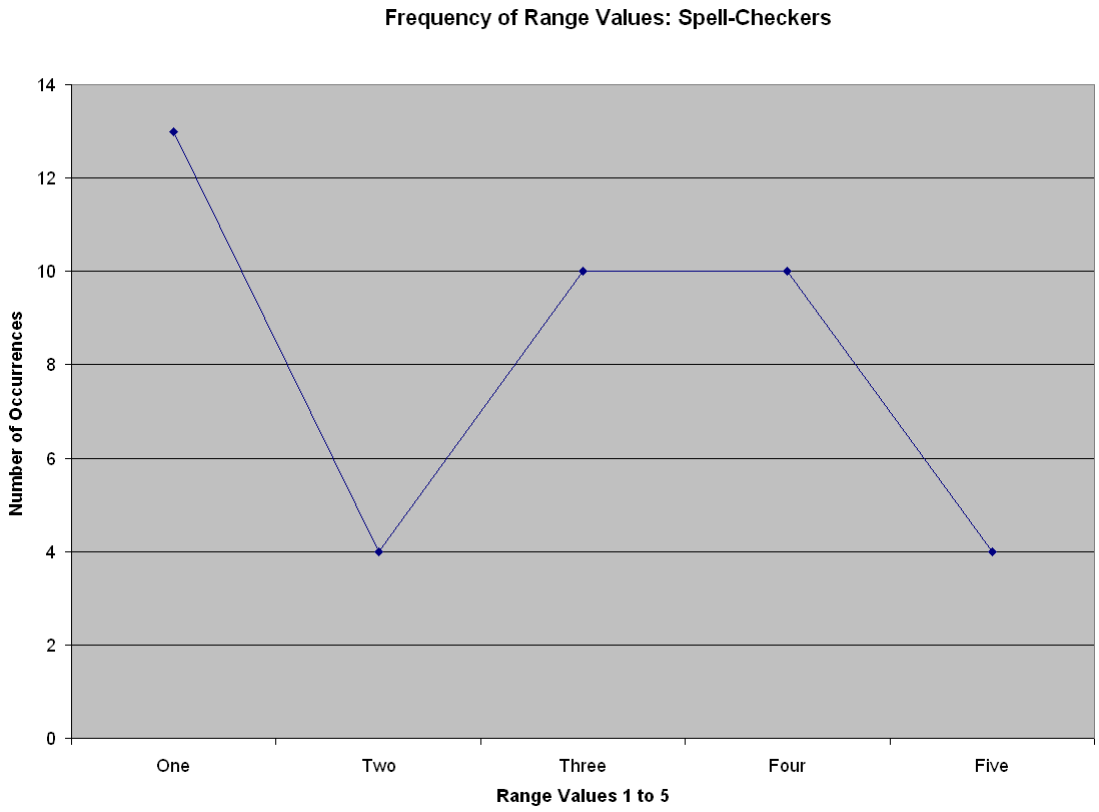


Figure 3.18: Spell checkers

3.3.10 Rating the contributing factors

In this section three of the range values will be examined with respect to how they relate to each of the predefined contributing factors. The rationale behind this is to observe the frequency of these three range values relative to each factor, and to subsequently identify which of the factors score the highest or the lowest according to the scaled ratings.

All the contributing factors shown in Figure 3.3 (nine of them) were considered in relation to the three range values: 1, 4 and 5. The motivation behind this is to identify which factors were considered as large contributing factors to the decline of society's spelling skills, somewhat contributing or not contributing at all. Figure 3.19 shows the trend for range value one in relation to the nine factors. Notably, the use of *word processing software* was deemed having the least impact on spell skill deterioration. Television came second.

Figure 3.20 shows the trend for range value four with respect to the same nine factors. Range value 4 has the highest frequency on this chart, and it would appear that *mixing languages* is identified as a factor that contributes extensively to the way spelling skills deteriorate. Coming a close second is *the music listened to,* especially the kwaito/rap genre.

Ratings of the contributing factors to spelling deterioration: Range value one on the scale of one to five for rating the factors

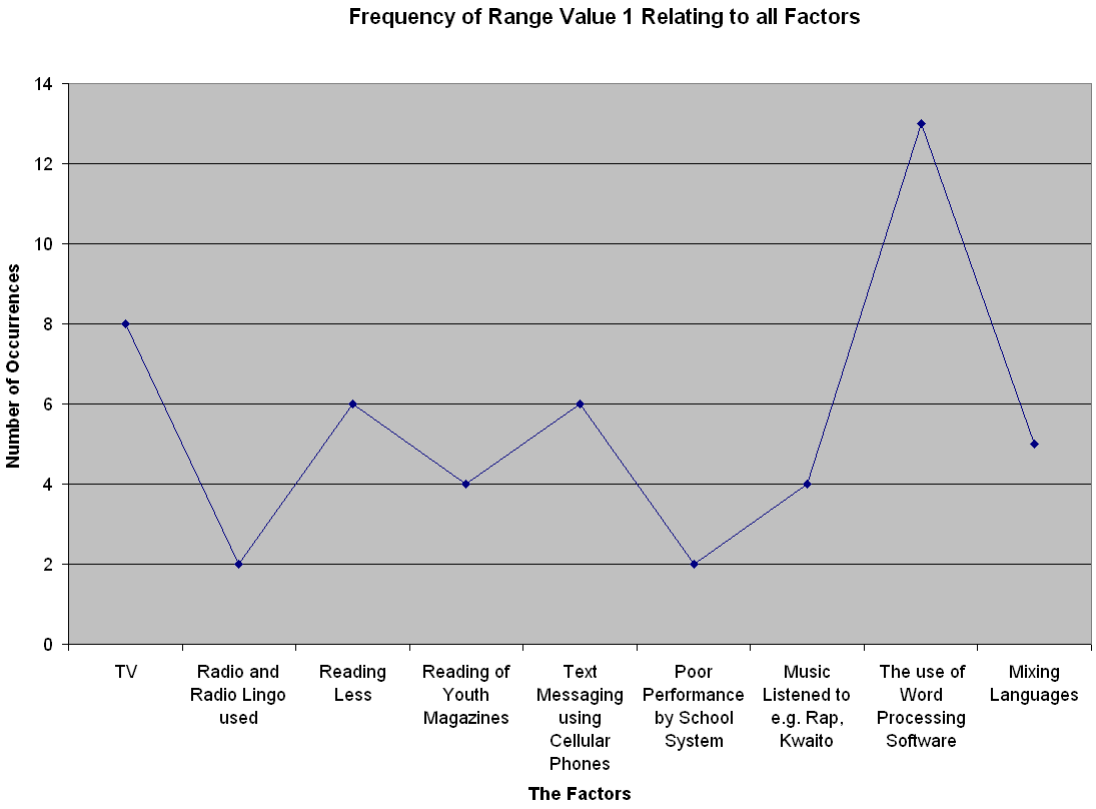


Figure 3.19: Value one

Ratings of the contributing factors to spelling deterioration: Range value four on the scale of one to five for rating the factors

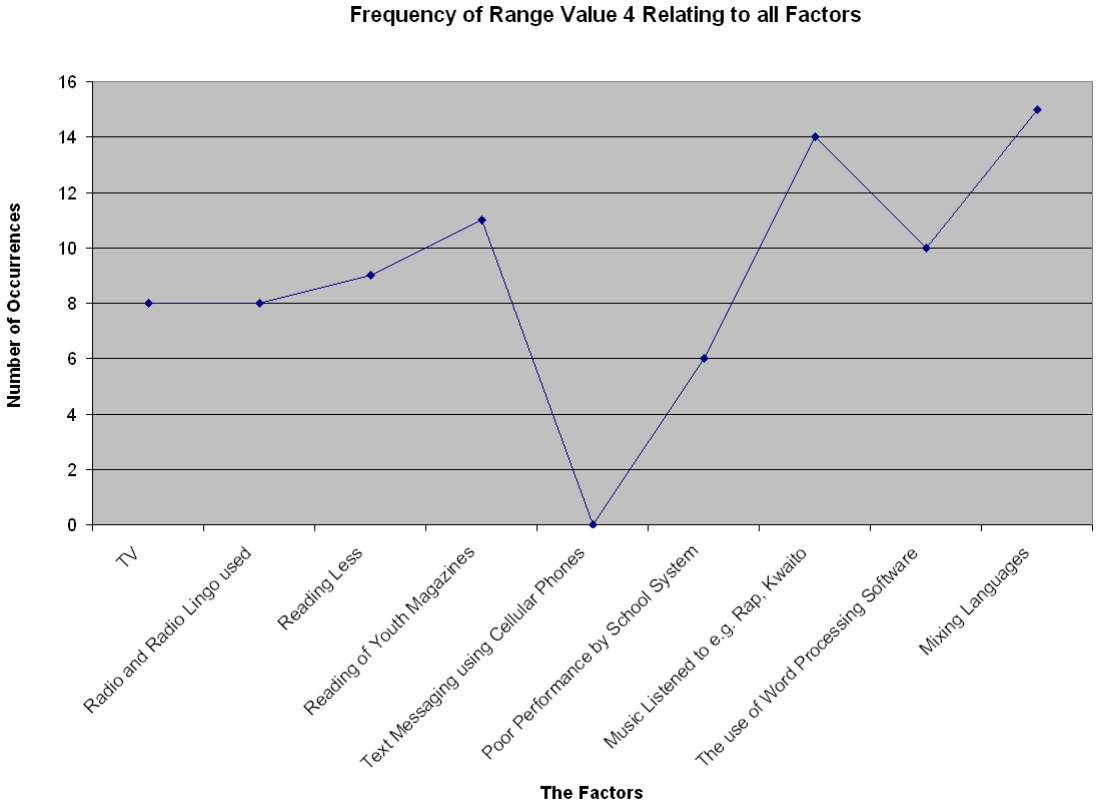


Figure 3.20: Value four

Figure 3.21 shows the comparison for range value five with respect to the same nine factors. As expected, if Figure 3.21 is compared with Figure 3.17, *text messaging* is identified as having a large contributing factor, when it comes to coupling a contribution factor with the range value vis-à-vis spell skill deterioration. If Figure 3.21 is examined and the charts earlier in this chapter are considered, it comes as no surprise that *music listened to* comes in second, followed by the factor *reading less*.

Ratings of the contributing factors to spelling deterioration: Range value five on the scale of one to five for rating the factors

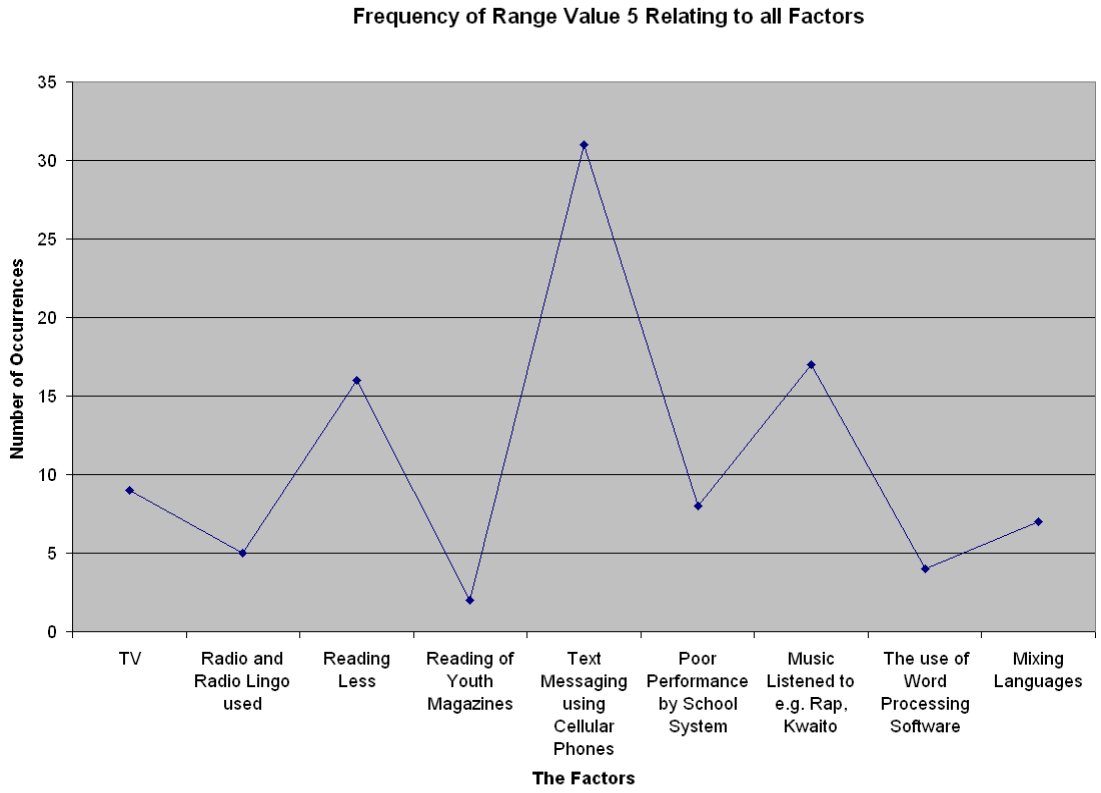


Figure 3.21: Value five

All the occurrences of range values, except the occurrences of range value one, were aggregated for each factor discussed. The occurrences of range value one were omitted because a range value of one meant that a factor with this value does not influence the way spelling skills have declined. The totals of the occurrences for each factor are depicted in Figure 3.22. This was done in order to try and identify the factors with the least and most impact on the deterioration of spelling skills.

When Figures 3.21 and 3.22 are compared, Figure 3.22 supports the view that text *messaging* had the largest total, supporting the perception that it has contributed largely to the way spelling skills decline. On the other end of the scale, as in Figure 3.22, the *use of word processing software* was deemed as having the smallest impact on spelling skills deterioration.

Total of range value occurrence for each factor

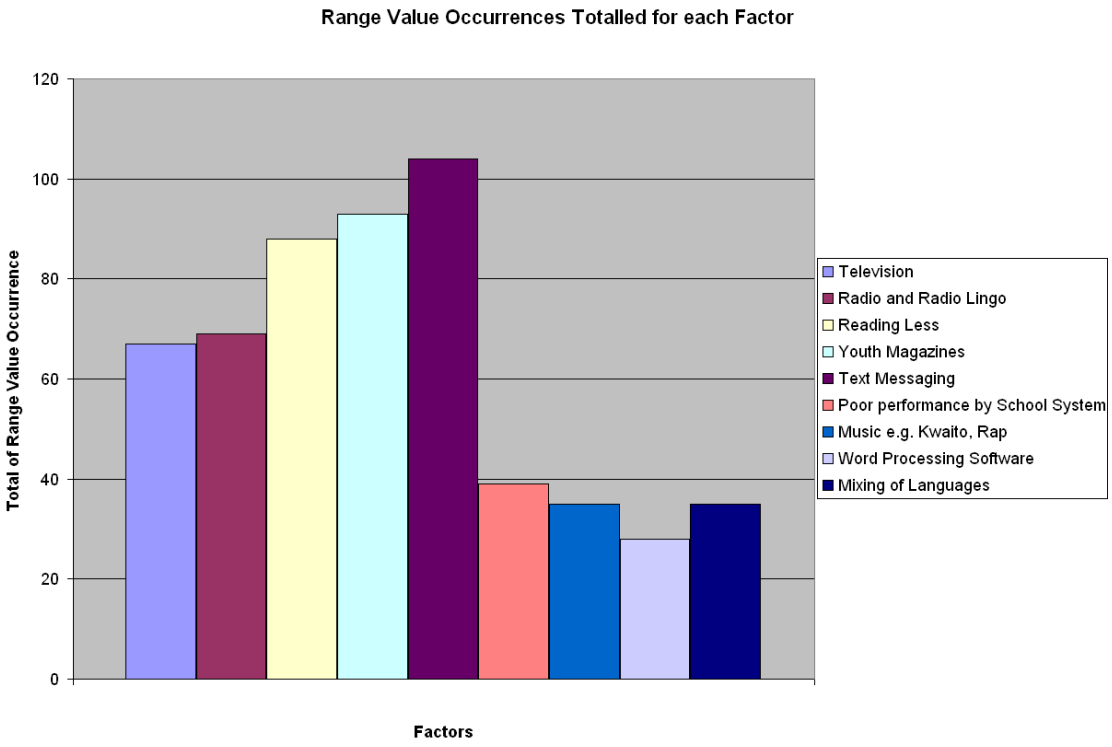


Figure 3.22: Range value occurrences

3.3.11 Spelling skills of the focus group

As mentioned earlier, twenty intentional spelling mistakes were made throughout the questionnaire, ranging from general spelling mistakes to using "*sms language*" in some cases. The final question posed asked the members of the focus group if they noticed any spelling mistakes whilst completing the questionnaire. Out of the forty participants involved, 75% (30 out of 40) replied "*yes*" and 25% replied "*no*".

Since humans read whole words and not letter-by-letter and assuming that the first and last letters of a word are in place; the word "*aoippntmetns*", for example, may still be read as "*appointments*". The focus group was then challenged with the task of finding the mistakes, not revealing how many or where they were, and replacing them with the correctly spelled version of the word. There were indeed twenty identical mistakes in the forty questions of

each questionnaire; hence any respondent who did very well would have identified all twenty of them.

Number of students who noticed spelling mistakes

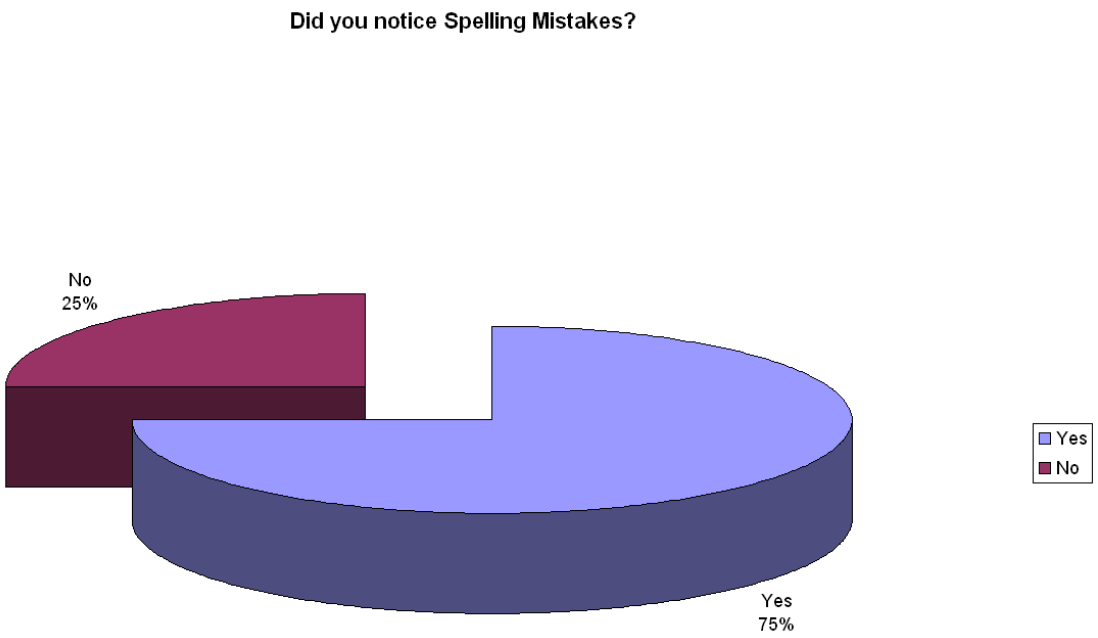


Figure 3.23: Amount of students who noticed spelling mistakes

The fact is that only 132 mistakes in total (if every participant identified every mistake, 800 mistakes would have been identified in total) were identified. This translates to 16.5% of the total number of mistakes. Figure 3.24 shows the results that were obtained, which were very alarming for this focus group, since the majority of them could clearly not identify most of the mistakes. In Figure 3.24, the researcher has clustered the number of participants versus the number of spelling mistakes identified. For example, 22.5% of the focus group could identify up to seven mistakes, another 7.5% could identify more than seven and up to ten mistakes and 12.5% could identify more than ten mistakes, but not more than fifteen. None of the participants could identify more than fifteen mistakes. This does, however, mean that at least 42.5% of the focus group could identify up to seven spelling mistakes.

Subsequently, the researcher took the 132 identified mistakes and counted the number of correct corrections as depicted in Figure 3.25. Only thirty-

seven (28%) of the 132 identified spelling mistakes were corrected properly and the remaining ninety five (72%) of the 132 spelling mistakes identified were incorrectly corrected.

Participants vs. Mistakes identified

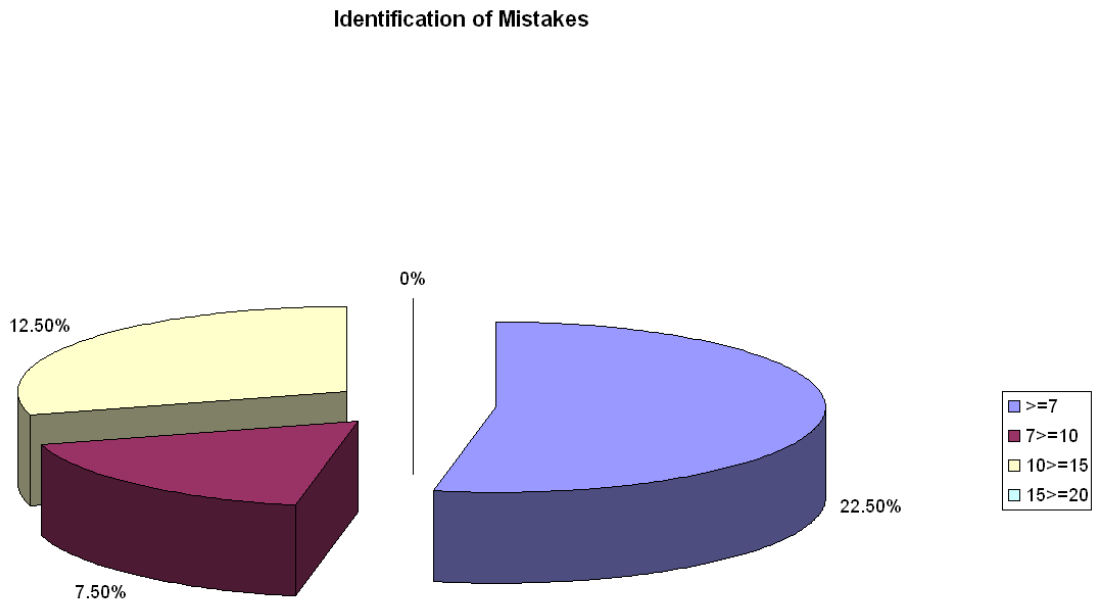


Figure 3.24: Number of participants vs. Mistakes identified

Corrections correctly made vs. Corrections incorrectly made

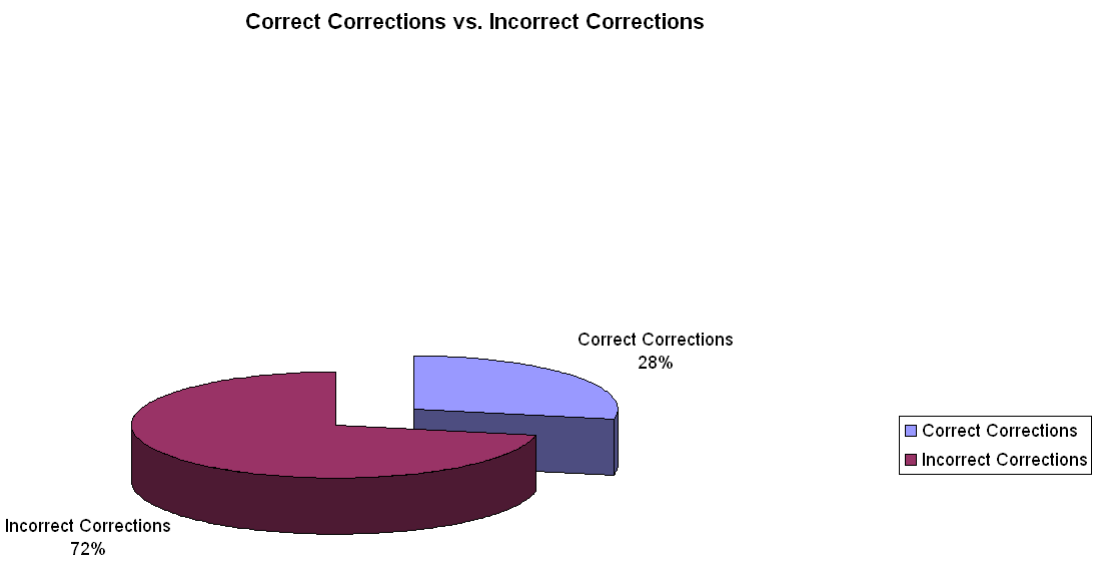


Figure 3.25: Correctly corrected vs. Incorrectly corrected

When considering all the extracted statistics based on the focus group's opinion in this chapter, we arrived at the following conclusion: spelling skills are not what they used to be, in any language, but especially in the language focussed on for this study, South Sotho. Spelling skills are definitely deteriorating. The *music we listen to*, *mixing our languages* as well as the fact that people *read less* and the use of *text messaging* as a means of communication all contribute to this fact. Finally, the researcher came to the conclusion that the average 20 year old university student, who had South Sotho as a subject during his or her secondary school career, struggles to spell efficiently in that language.

The implementation and execution of the questionnaire for information discovery purposes was very fruitful. The results obtained satisfied all four of the initial purposes of why this study was undertaken as discussed in section 3.1.

Firstly, it seems that the state of spelling skills have indeed deteriorated, especially among the focus group and as depicted in this chapter, various factors have contributed to this fact. Extra features to the already vast array

of features of commonly available spell checkers were identified by the focus group. Lastly, the researcher concludes that there is considerable evidence to support the development of a spell checker and corrector for South Sotho.

Chapter four explores related work done in the field of text-based spellchecking and correction. Also discussed are algorithms developed by other researchers with regard to spellchecking, correction and suggestion as well as techniques for word-error correction.

Chapter 4

4. RELATED WORK

This chapter investigates work that has already been done in this field, not specific to a language, so as to disseminate the algorithms and checking techniques used by different researchers and programmers and the constraints of spell checkers and correctors. The chapter also explores why spelling mistakes occur and how an electronic dictionary's size can be reduced.

4.1 Related Work

4.1.1 Applications to spell-correction

As mentioned in chapter two, two types of spelling programs exist: *spell checkers* and *spell correctors*. The job is easier for a spell checker: Input a file of text and identify the incorrectly spelled words [15]. A spell corrector, on the other hand, can detect misspelled words; it tries to find the most likely correct word and performs automatic correction of the misspelled word.

Spelling-correction is an important application that forms part of error-tolerant recognition [13]. Most techniques assume a wordlist or the use of a digital or electronic dictionary of all the words in a specific language. Different checking methods are used to identify and correct inaccurately spelled words within this framework. These approaches are suitable for languages like English, for which it is possible to enumerate such a list. They are not directly suitable or applicable to languages, like German, which have very productive compounding, or agglutinative languages like Finnish, Hungarian etc.

When it comes to the general architecture of a spell checker and corrector, Van Huyssteen notes that such an application should mainly consist of two kinds of modules: A *look-up* module and a *rule-based* module, used for morphological analysis [42]. Software reusability was one of the main motivators behind the mentioned architecture.

Van Huyssteen and Van Zaanen [40] implemented two look-up modules in their work; a simple lexicon look-up module and an error-detection module. The lexicon (dictionary) look-up module uses a size increased lexicon, mainly for two reasons:

- "in order to gain a few percentage points on lexical recall", and
- "to intercept problems that might not arise during morphological analysis" [42].

A question now arises: Will the larger lexicon not impact the performance of such a program adversely? The answer is a definite "no", not if the application is run on newer machines. With the current size and speed of computer hardware, an increased lexicon should not have a negative impact on performance. The effects of the size increase are debatable when older machine platforms are considered.

The error-detection module consists of a look-up section, similar to that of the look-up module, but with the difference that it contains only a list of frequently misspelled words, and a section where errors are detected based on an analysis method developed on a 4-gram analysis for Afrikaans [41]. One can only assume that this module is invoked first when it comes to spellchecking.

The look-up section contains a small body of errors and when the look-up procedure starts, misspelled words with the correctly spelled form of that word, are sent directly to a suggestion module [40] [41].

Some recent approaches to spelling correction have used morphological analysis techniques [17]. All inflected word-forms of languages like English can be included in a word list, which can be used to construct a finite-state recognizer structured as a standard letter-tree recognizer. Error-tolerant recognition can then be applied to this finite-state recognizer [11]. This method could also be applied to a language such as South Sotho (which is not a highly inflective language), where e.g. plural forms consist of a stem, or root, and a prefix (for example “di”). In some cases, this proposed approach may not be efficient and may be augmented with language-specific heuristics. For instance, in spelling-correction, users usually replace non-ASCII characters with their nearest ASCII equivalents when using non-standard keyboards, or using a number of keystrokes to input non-ASCII values.

Yet another approach proposes that common misspellings for (a) particular word(s) be included in the dictionary or word list as well [15]. This means that a word, for example, *greatfully* would be tagged as a misspelling of *gratefully* and entered as a known common misspelling of the word. The problem with this approach is the lack of a source of known misspellings and the frequency of even the most common misspellings. Studies conducted by Kukich [21] reported that 80% of misspelled words contain a single error of one of the unit operations, although in specific applications the percentage of such errors is lower.

Peterson of the University of Texas at Austin created a machine independent spell checker that can check and correct spelling mistakes made in the American-English context [15] using an input file.

The programme uses a three-level dictionary of common English words. Firstly it consists of a small table of most commonly used words in the English language; Secondly, a table of words containing all of the words already in the document to be checked as well as, the secondary, static, disk-based, large dictionary. The second component of the three-levels is composed of a hash table containing the

document words. A hash table is a random access device that is used for looking up a word by a type of code, value or key. These codes, keys and values are derived from characters in the word itself [22]. For reasons of brevity, we only mention two types of hashing, namely *partial hashing* and *total hashing*, which are used to construct the so-called hash tables.

Peterson's application works on the principles of building a list of all the tokens found to be distinct in the document or input file. Each token is then looked up in the dictionary. If a token is found in the dictionary, the assumption is made that the word is spelled correctly; otherwise it is assumed that the word is spelled incorrectly. All tokens (words) that have not been found in the dictionary are then printed to the screen. With today's interactive word processing software, one could surely improve on Peterson's model, but it would serve no purpose with regard to this research project.

Thomas N. Turba's spell checker consists of the following fragments [22]: *Lexical analyzers* that consist of different rules for recognizing words in the document as well as forming spelling suggestions. The output of these analyzers is sent to the next part of his architecture, the *Word Checker*. This segment of the application strips the words from prefixes and suffixes and checks in which part of the dictionary the word falls. If it is not found, it is considered an error and different forms of the word concatenated with its pre- and suffix are tested.

Thirdly, the programme incorporates a *Master Dictionary* that is found in secondary storage and is in a special form. A *Cache*, which is a random access list of words from the master dictionary, is implemented as a hash table (note that Peterson also used a hash table), but also as a tree that contains word entries.

The fifth part of the application is a *High Frequency Dictionary* which consists of frequently misspelled words. Van Huyssteen [41] [42] also mentioned using a

similar structure in the look-up section of his checker based on an online spelling competition.

Finally, a *Stripping Dictionary* is implemented. It contains prefixes and suffixes for stripping, much as the design employed in this study, which will be explained in the next chapter. Strangely enough, Turba did not implement stripping in his final implementation [22]. For him it was a matter of design philosophy. Further study into the South Sotho language is definitely necessary. Some related words might not have the same stem or root, posing a new problem in terms of error-correction and internal linking of the words (links such as pointers may be considered to help solve this problem), or, two adjacent words in the South Sotho language might have one meaning or be used to express or convey a single idea.

4.2 Reasons why words are misspelled

According to studies done by Damerau [5], most spelling mistakes are the result of:

- transposition of two letters
- one extra letter
- one letter missing
- one letter wrong

These rules can be used as the basis for an algorithm to indicate misspelled words and indicate or suggest correct form(s) of a word. A spelling-corrector called *DEC 10* [5] uses a method based on these rules and will be discussed later.

A spelling-error is introduced in various ways and the following are deemed to be the most important [3]:

- *Author ignorance*: Such errors can lead to consistent misspellings. The problem occurs because of the difference in how a word sounds and how it is actually spelled.
- *Typographical typing errors*: These are less consistent but more predictable, the reason being that they are related to the position of keys on the keyboard and probably result from errors in finger movements.
- *Transmission and storage errors*: This refers to the relationship between the specific encoding and transmission mechanisms. Earlier work on spelling correction actually focused on the problem of recognizing OCR input and the recognition of Morse code [36].

The above mentioned facts mean that there are obviously different sources of errors, which can only mean that there are some algorithms that will work better on some errors than on others.

4.3 Checking methods

The methods described briefly below have been used on their own or, in some cases, have been combined in certain algorithms in order to enhance performance. The advantages and disadvantages of each are also discussed.

4.3.1 Statistical analysis

This method uses frequency analysis of neighboring characters in a word to be checked. To accomplish this, a sample text is used from which frequency counts are derived for fixed length character groupings like two or three character groupings (digraphs or trigraphs) [42]. Once the frequency table has been built, it can be used to check text. If a word has a low frequency profile, it can be flagged as a possible spelling error.

The *primary advantage* of this method is that it tries to reduce the amount of storage needed to verify a word, and, it can check a large number of words. The *disadvantage* of this method is that a large dictionary must be used with the frequency table to avoid validly spelled words being flagged as incorrect and a large number of words that are incorrectly spelled not being flagged.

4.3.2 Stripping

This method strips a word from its prefix and suffix, which then produces its root [22]. The root is then checked in the dictionary. The *advantage* of stripping lies in the fact that a small dictionary can be used to check a large number of variations of a single word. Flexibility is also one of this method's strong suits, because variations of a single word do not have to be explicitly stored to be recognized.

Flexibility also contributes to the *disadvantage* of this method. The reason for this is that a valid pre- or suffix combined with a valid root is not sufficient criterion for a word to be correctly spelled, e.g. the word *misspelled* might be incorrectly spelled as *dispelled*, which will look correct to the checker, but, in reality, an invalid prefix has been used; which provides the word with a completely different meaning.

4.3.3 Complete look-up

When implementing this method, the use of a large dictionary is a necessity [28]. The dictionary must contain all forms of valid words that need to be considered. The *advantage* of complete lookup is that only valid spellings will be accepted and the rest will be flagged as erroneous.

Disadvantages include increased look-up time, the need for larger storage capacity for the dictionary as well as increased complexity. As mentioned

previously, computing power has increased exponentially as did storage capacity. However, the former disadvantage still has merit as it does indeed need more storage space and processing resources, although the impact on performance will be less severe than in the past.

4.4 Techniques for word-error correction

Kukich [21] has long been renowned as one of the authorities when it comes to spell checkers and spell correctors. She identified three problems with regard to correcting words that have been spelled incorrectly and also subsequent techniques to overcome these problems. The problems are listed as:

- Detection of the errors
- Generation of candidate corrections
- Ranking of the suggested candidate corrections [12]

The above mentioned problems are usually tackled by using techniques that confront each problem as a separate process. With the exclusion of n-gram (also used by Van Huyssteen [41] as 4-gram analysis) analysis that found its way into spellchecking, Kukich [21] organized correction techniques into six main classes:

- Minimum edit distance techniques
- Similarity key techniques
- Rule-based techniques
- N-gram-based techniques
- Probabilistic techniques
- Neural nets

4.4.1 Minimum edit distance techniques

This is by far the most studied of the spell correction algorithms. These algorithms revolve around the principle that a minimum edit distance between a dictionary entry and the misspelled string should be computed. This refers to the minimum number of times a word has to undergo editing operations, for example, insertions, deletions etcetera, to transform one string to another [12].

4.4.2 Similarity key techniques

These techniques focus on mapping every string into a value or key so that strings that are spelled similarly will have identical values or keys. This means that when a value is calculated for a string that is misspelled, it will provide a pointer to the similarly spelled words in the lexicon (dictionary). One of the similarity key techniques' advantages is speed, the reason being that the misspelled word does not have to be checked against every word in the dictionary [12]. A more detailed explanation regarding similarity keys is discussed in the next chapter.

4.4.3 Rule-based techniques

Algorithms or heuristic programmes that represent, or rather, attempt to represent, a certain knowledge of common patterns related to spell-errors, in the form of rules to transform misspelled words into validly spelled words, can be seen as rule-based techniques for word-error correction.

Numerical scores are assigned to each candidate based on a predefined probability estimate for each, i.e. what are the chances that a particular error will occur. The candidates for correctness are then ranked according to these numerical values [12].

4.4.4 N-gram-based techniques

N-grams including trigrams, bi-grams and unigrams have been used in various spellchecking and text recognition practices. One of its first applications was with regard to OCR. It was used in this context to capture lexical syntax of dictionaries as well as to suggest legal corrections in applicable documents. It has also been applied in conjunction with vector distance measures to locate and rank candidate corrections and represent the words and misspellings as vectors. In spell correctors it is used to locate candidate corrections in a dictionary by acting as access keys into the dictionary.

N-gram-based techniques can be applied to perform or execute different processes, for example, to retrieve candidate words for a misspelled word, to detect the physical error made, to do similarity ranking, etcetera [12].

4.4.5 Probabilistic techniques

Probabilistic techniques are products of n-gram-based techniques, in both paradigms of spell correction and text recognition [32]. According to Kukich [21], two probabilities have been exploited: *transition probabilities* and *confusion probabilities*. The former works on the probability that one letter will be followed by another given letter, thus, transition probabilities are language dependent. These probabilities can be estimated by collecting frequency statistics using the method described in 4.4.4, namely, n-gram-based techniques. The statistics are collected from a large corpus, or body of text.

Confusion probabilities refer to an estimated number of times or the frequency a provided letter is substituted or even mistaken for another letter. Confusion probabilities are dependent on the source of the text or the device used, for example, different OCR devices will have different probabilities when it comes to confusion probabilities [12].

4.4.6 Neural Net Techniques

Neural nets are such likely contenders for spell correctors because they have the innate ability to perform something called associative recall based on input that is not complete. Neural nets can be trained, which means that they can potentially adapt to error patterns that make themselves apparent. The adaptation process is facilitated by algorithms, first introduced by Rumelhart [37], which became the most widely used algorithm to train neural nets. Neural nets have the abilities to maximize error-detection and correction within a certain community of users, where similar mistakes are made and the neural net learns the mistakes made as well as the correct counter measures to be taken and suggestions to be made [12].

4.4.7 Other techniques

Golding [9] presents an additional five methods for spell correction when word context is considered. They are outlined below:

- *Baseline method*: Acts as an indicator of the so-called “minimal competency” in regard to comparison to other methods.
- *Context words method*: This method tests for specific words as candidates of correction within a word-range of the misspelled word.
- *Collocations method*: This method tests for syntax patterns in regard to the misspelled word or any word identified as a target (within the area of corpus linguistics, collocation is defined as a sequence of words or terms which co-occur more often than would be expected by chance [25]).
- *Decision lists*: A decision list is constructed based on collocations and context words.
- *Bayesian classifiers*: Bayesian classifiers are constructed using context words and collocations. A Bayesian classifier is a classifier based on Bayes’ theory: Bayes' theorem (also known as Bayes' rule or Bayes' law)

is a result in probability theory that relates conditional probabilities. If *A* and *B* denote two events, *P* (*A*|*B*) denotes the conditional probability of *A* occurring, given that *B* occurs. The two conditional probabilities *P* (*A*|*B*) and *P* (*B*|*A*) are different in general. Bayes' theorem gives a relation between *P* (*A*|*B*) and *P* (*B*|*A*) [25].

The last two methods outlined above are hybrid methods based on the combination of context words and collocations.

4.5 Obtaining a dictionary

All spell checkers and correctors use a dictionary of some kind, and many dictionary representations exist. Dictionaries can be both static and dynamic. The most difficult task in developing a spell checker is obtaining the right dictionary [21]. The South Sotho language may have a lexicon size of up to 58 000 [41].

The task of obtaining a dictionary may not seem a difficult one, for example, the researcher could have approached a company that produces dictionaries and purchased one. However, there are numerous problems with regard to acquiring a dictionary as indicated below [21]:

- Most companies that produce dictionaries are reluctant to sell a machine readable copy of a dictionary, even if it is stripped of all meanings and consists only of headwords. Some companies will sell it at a substantial price and/or charge royalty fees on its use.
- If such a dictionary from a publisher could be obtained, it would not contain all forms of a word, and most notable of the missing words will be plurals, but a large number of other forms and application-specific terms will be missing.

- A general dictionary will contain many similar words that are not suitable for some applications. For example, in a business environment, *cheap* might be an acceptable word, whereas *cheep* would not.
- Although it might seem odd, dictionaries obtained from a publisher would often also contain spelling errors, though the percentage would be low.

If a dictionary is not available, word verification is a little more difficult but not impossible. Frequency counts can be taken as an approach. If a certain word appears many times in a document and for a large percentage of its appearance it is spelled consistently, it can be assumed by the software that it is probably spelled correctly. For this method to have a high degree of success, ten to twenty documents should be used to build the frequency table out of word-counts.

4.6 Reducing the dictionary size, the use of tokens and a useful algorithm

The performance of a spelling checker/corrector is very important. Batch checking of words is not an effective solution to follow; a program must rather check each occurrence of the word in the order in which it is used. The structure of the dictionary used is thus very important. Factors that influence the way in which to structure a dictionary include memory size, file access methods, the existence of virtual memory, etc.

Building tables of words such as shown in Figure 4.1 could improve search time as well as contribute to a reduction in memory used. According to research done [11], three tables can be built for each specific document from a dictionary. Extracted words from a dictionary are then placed in the table using tokens.

The first table is built from the most commonly used words in the document, determined by the number of distinct tokens (each word is assigned a token) observed. The first small table thus consists of words/tokens which occur

frequently in that document. Secondly, a table of words is build which have already been used in the specific document.

Finally, the large list of the remaining words in the main or complete dictionary is placed in the third table. If a word is found in this level, it is moved to the second table.

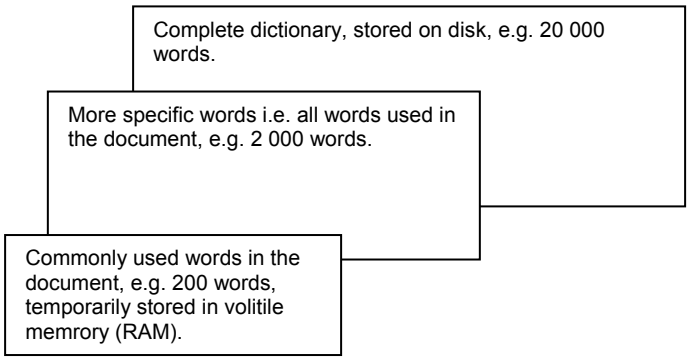


Figure 4.1: Possible Sizes and use of three separate dictionary-tables

The following is an algorithm, used in a spelling corrector named *DEC 10* [43]:

> *For each token which is not found in the dictionary, construct a list of all words which could produce this token by one of the rules mentioned in section 4.3.*
>
> *If the list has exactly one candidate, guess that word and ask the user if that word is the correct spelling.*
>
> *If the list contains several words, state this fact. The user may then request the list and select one as the correct word, or indicate one of the normal options of replace, replace and remember, accept, accept and remember, or edit.*

The candidate list is formed by multiple searches of the dictionary. Transpositions can be detected by transposing each pair of adjacent characters in the token, one at a time, and searching for the resulting token as shown in Figure 4.2 below [5]:

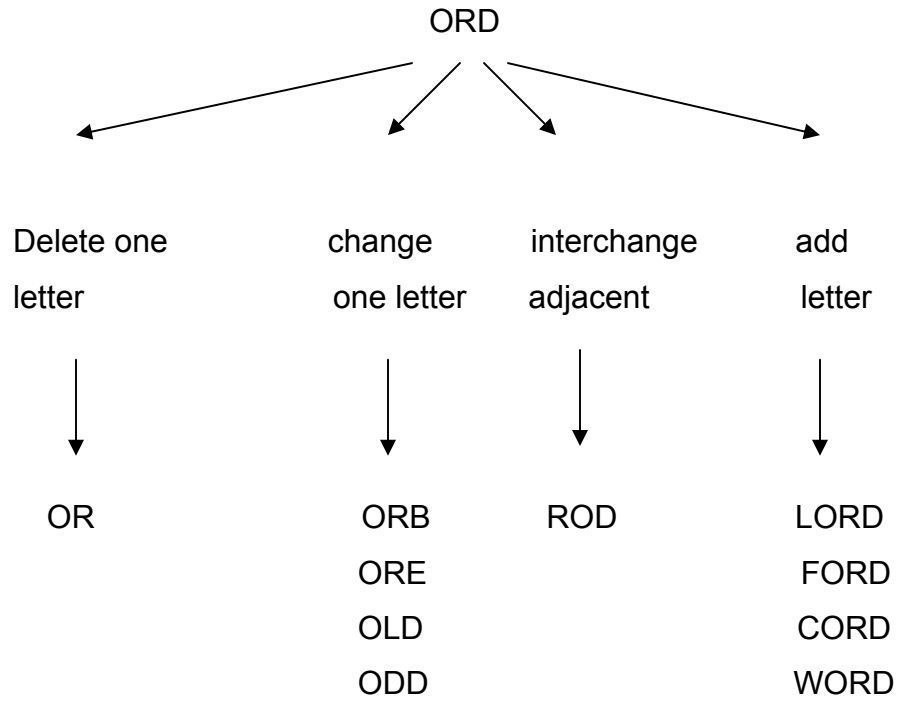


Figure 4.2: Illustrating an incorrectly spelled token and correctly spelled tokens when the rules discussed above are applied

The Slavic languages consist of millions of word forms which differ in the use of prefixes and suffixes of various forms. The work done by Hajič and Drózd [13], solved the problem of creating a spelling checker and corrector for Slavic languages in general and Czech in particular. They used methods to compress the word forms and linguistic knowledge about the regularities of the morphological behavior of the words to overcome this problem. In their basic model of inflection they assumed that a word form consists of a concatenation of a stem and an ending.

To use this model in the software, they had to define both *stem* and *ending*. “Stem” meant, for them, the part of the word which does not change in the course of inflection, and the term “ending” means the part of the word form which, when appended to the stem, completes the stem to a meaningful form.

The system also provided functionality of adding word forms to the user’s own diary, like e.g. a name of a friend that is not recognized by the software as a correctly spelled word form. The programmers then added automation to the updating of the dictionary used, by asking the users a series of questions when adding a word, e.g. is the word a noun, verb, adjective etc. to add it to the correct class.

Finally the software was implemented with a memory requirement of 400K. It was able to check one screen of a 60 column standard text (approx 200 words) in 3 seconds on a 10MHz PC. The dictionary used covered 80 000 to 100 000 Czech words with 7 000 of the most frequently used Czech words loaded into the memory. Since October 1989 the system has been available to anybody wishing to avoid misprints when writing in Czech using a computer.

4.7 Existing algorithms

The following two algorithms were selected for discussion because they have been implemented with success in related work. A short discussion of both follows:

4.7.1 The Speedcop correction algorithm

The Speedcop algorithm [28] generates a similarity key for each word in a digital dictionary. It then sorts these keys in key order. Misspelled words are corrected by locating words whose keys most closely collate to the key of the misspelled word and then selecting possible candidate words from these.

Similarity keys exhibit less scatter due to the fact that a random number of strings may share the same key when compared to the original strings. The similarity of the original string is measured by ordering the strings' features in an order so that the distance of collations between two keys serves as the measure. The key generated from the misspelled word is compared with the dictionary keys by using a random access method. The center of the similarity-ordered set of dictionary words is calculated by using the last key to compare keys equal to or fewer than the misspelled word's key and assigning that as the center. By reversing the error operation encountered, each member of the set of retrieved, similar words is tested as a correct candidate for the misspelled word [17]. This idea was also proposed by Damerau [7]. This algorithm does not take word context into account.

4.7.2 Longest string-matching algorithm

This algorithm was implemented by Van Huyssteen and Van Zaanen [41] to serve as an effective way to identify word boundaries.

The principle behind the algorithm is to search for the longest part of a word (from left to right) that is an element of the spell checkers dictionary, or lexicon.

The algorithm starts by looking for a valid suffix by evaluating the most right hand side of the word. The algorithm is then applied to the remaining part of the word, trying to match a valid prefix. After the above mentioned has been performed, an element in the spell checker's word list should remain. This word is then checked against the lexicon and candidate words for correction are suggested (if the stripped word is incorrectly spelled), by using a machine learning approach; this is done by implementing decision trees and converting words into feature vectors [42].

The researcher decided to use a combination of stripping, the development of hash code values for word-storage and a degree of complete lookup with regard to the hash code values to develop word-checking and misspelled word identification algorithms.

The researcher used the principles discussed with regard to similarity keys and the longest string algorithm for algorithm development.

Chapter five focuses on the checking, identification and suggestion algorithms developed by the researcher, the implementation of these algorithms, the results yielded by the algorithms as well as their performance.

Chapter 5

5. DEVELOPING eSPELLINGPRO SA SESOTHO SA LEBOA

The development of the software application, *eSpellingPro sa Sesotho sa Leboa,* was a challenging undertaking in its own right and the final system is the culmination of nineteen months worth of research and work. While this project was in its development stages and right through to the end, many challenges that had to be overcome in order to prove the study hypothesis as discussed in chapter two.

These challenges included:

- Using a multi-threaded execution methodology to ensure optimum CPU usage.
- Capturing each word and validating the spelling of the word captured as apposed to validating an entire piece of text for misspelled words.
- Identifying misspelled words and flagging them as misspelled whilst not flagging correctly spelled words as erroneous.
- Facilitating a degree of automatic spell correction of incorrectly spelled words within the software.
- Enabling the software to make valid suggestions with regard to the misspelled words identified.
- Providing the user with the Afrikaans and English meanings of a South Sotho word.
- Ensuring software performance within the metrics established for the software in chapter two, that is, validating the typed word, identifying and flagging errors that occurred and suggesting possible corrections vis-à-vis the flagged word. All this within 600 milliseconds and all with a high degree of efficacy.

This chapter focuses on how these challenges were overcome and explains the algorithms developed to achieve the former and gives the results of software performance tests that were conducted.

5.1 Threading

When one examines a car, it performs a great variety of processes in parallel, or rather, concurrently. The pistons move up-and-down, fuel is fed to the combustion chamber, valves open-and-close whilst power is transferred to the wheels, which on their part turn, and so forth. All of these happen at the same time.

Computers also encompass the ability to perform operations simultaneously. Unfortunately, there are many programmers who do not exploit this advantage as there are many programming languages in use that do not allow a programmer to specify concurrent actions to be executed. These programming languages only enable the programmer to create a software application where the different operations follow sequentially.

Visual Basic.Net was used as the development environment and language for creating *eSpellingPro sa Sesotho sa Leboa*. Visual Basic.Net enabled us to specify “threads” of execution within the programme. Each “thread” contains a portion of the programme that can execute concurrently with other “threads”, which in actual fact means that the processing resources are used to a greater capacity than with sequential programme execution. This capability is called *multi-threading*, because different program segments run on multiple processing “threads”.

Employing this feature of the programming language has contributed to the software performing within the established parameters.

5.2 Capturing words

In a spell checker and corrector, the words that are typed in the text corpus are arguably the most important. The researcher's objective was to capture words one word at a time in contrast with some software that allows the user to type a large portion of text, even a whole document, and then check the entire potion of text for incorrectly spelled words via the click of a button. By checking the words one word at a time meant that the user could be alerted to a misspelled word immediately after the mistake had been made. Where automatic correction could be applied, the word could immediately be automatically corrected and time would be saved because the operation of checking one word against the dictionary would have been much faster than checking, for example, a portion of text containing one hundred words.

We settled on the decision that capturing words, one at a time, meant that the program had to capture each letter of the word as it was typed. The delimiter that has been utilized, signifying that a complete word was typed, was the use of the spacebar. When a user presses the spacebar, the programme realizes that one complete word has been typed and captures it as such.

The procedure of capturing one character at a time proved to be a greater challenge than initially thought. Let us assume, hypothetically, that the user wanted to type in the word "paate", but, instead, entered the word "paete". Upon making this mistake, the user, before pressing the spacebar button, realizes his or her mistake and corrects it him or herself. The user presses the backspace button continuously until he or she reaches the letter he or she wants to replace, then replaces the "e" with an "a" and completes the word to what the user originally intended to type.

Because of the binary internal system representation of characters, by pressing the backspace button, the programme captured the character that represents the backspace button on the keyboard. Since the programme captures each word

on a character-by-character basis, the system would have incorrectly flagged the user-corrected word “paate”, as misspelled.

The reason for this is that, although the word “paate” appears on screen, internally, the word would have been captured as “paete●●●ate”, the “●” representing the backspace button’s character representation that was captured each time the backspace button was pressed. This meant that each and every keyboard button pressed would be captured by the software application and may have resulted in incorrect flags. In this case, which is a false negative, a valid word would not have been recognized by the software as valid, because the dictionary does not contain the word “paete●●●ate”, only “paate”. Figure 5.1 shows the above mentioned output via a message box.

Figure 5.1: Output of a captured word and the effect when the internal representation of certain keyboard keys is not handled by the software, in this case, the backspace key in particular

The delete key was also tested regarding its effects on the word-capture procedure, as was the shift, alt and ctrl keys. The results illustrated no effects on the structure of the word when it was captured with regard to these keys.

What would happen if the same scenario as above was used, but instead of using the backspace key from the very last character of the word to where the correction is intended, the mouse cursor was placed just after the first “e” in “paete”, before the backspace key on the keyboard was pressed to eliminate the “e”, replacing it with an “a” to correct the word?

In Figure 5.2 it is shown that the representation of the backspace character is again found at the end of the captured text as well as the character with which the user intended to replace the first occurrence of the letter “e” in “paete”.

Figure 5.2: The effects of the unhandled internal representation of the backspace key and its influence on an attempted correction by the user

These tests suggested that the backspace key’s internal representation character as well as any attempted insertion of a character would be added to the back of a captured word.

Figure 5.3 is a snippet of source code that illustrates how each character is captured one at a time and includes a solution to the problem vis-à-vis the capture of the internal character representation of, in this case, the backspace key.

As already mentioned in the text, one of the challenges was to prevent the application from capturing unwanted characters, like, for example, the character representation of the backspace button. The procedure in Figure 5.3 captures each character that has been pressed on the keyboard. The first if statement in the source code snippet of Figure 5.3 converts the character pressed on the keyboard to its hash code representation. The backspace button’s hash code representation is “524296”. When the programme encounters this hash code, it realizes that the backspace button has been pressed, and ignores it by using the second if statement, which trims the word from the backspace character that has been pressed.

```
Private Sub txtPage_KeyPress(ByVal sender As System.Object, _
ByVal pressedKey As System.Windows.Forms.KeyPressEventArgs) Handles txtPage.KeyPress

    If pressedKey.KeyChar.GetHashCode = 524296 Then
        'the hashcode value of the Backspace key is 524296
        If strWord <> " " Then
            strWord = Mid$(strWord, 1, strWord.Length - 1)
            'strWord is "trimmed" from the backspace character representation and
            'the characters that was erased from the word.  E.g. if the word "Peter"
            'was entered, and "er" are erased, the .KeyChar function captures two
            'backspace characters (represented as squares.  If the character "e" is,
            'for example, added, and the above was not done, strWord would look like
            'this: Peter--e is stead of e.g. "Pete Pan", where the "--" represents
            'two squares, which represents that the backspace key was pressed twice.
        End If
        Exit Sub
    Else
        chrCharPressed = Trim$(pressedKey.KeyChar)
        'the character that was pressed on the keyboard is captured.
        strWord = strWord & chrCharPressed
        'the word is captured character by character.
        intPressedCounter = intPressedCounter + 1
    End If

End Sub
```

Figure 5.3: Capturing each character one at a time code snippet

The function *Mid$* extracts an identified portion of a word and has three parameters. The first parameter represents the base string that the researcher wished to work with, which, in this case, was the first character or the concatenated results of characters typed in. The second parameter identifies the position from where, in the base string, the function should start extracting a portion of text. The third parameter signifies the length of the portion of text that we wished to extract, in this case, the length of the character string minus one.

If the objective is to capture every character up until the backspace button's character representation, all character up to that point in the base string should be extracted. To accomplish this, the function starts extracting the portion of characters from the first character (second argument) of the base string up until the second last character of the captured word (third argument). This proved to successfully remove the unwanted characters from the captured word.

When considering the scenario which relates to Figure 5.2, we suggest that a "replace" function be utilized where the deleted character is replaced with the newly typed character. The inclusion of details regarding such a function was not

deemed necessary for this study, as we already proved the efficacy of the employed extraction function in Figure 5.3 to demonstrate the software's ability to deal with the occurrence of such potential problems.

The fourth last line of code, in the code snippet of Figure 5.3, demonstrates the programme's ability to concatenate the letters that have been captured by the programme with the characters that have already been captured and assigns it to a variable that represents a full word. If it is assumed that the user desired to type the word "paate", the first character typed would be the character "p", which is assigned to the variable *strWord*. The next character captured would be "a', which is added to *strWord* that already contains the letter "p", and so forth.

The programme needs to be aware of each full word that was captured. We decided to use the spacebar as the identifier to signify that a full word was captured. Each time the spacebar is pressed on the keyboard, the concatenation of characters collected by the code in Figure 5.3 is extracted as a full word and sent to the segment of code with regard to the checking procedure.

Figure 5.4 illustrates a source code snippet. The procedure captures the key value that has been pressed and checks if it is indeed the spacebar. If it is the spacebar that has been pressed, the word is trimmed from any leading or succeeding white space characters and sent to the first checking module, which is explained later.

```
    Private Sub txtPage_KeyDown(ByVal sender As System.Object, _
    ByVal pressedKey As System.Windows.Forms.KeyEventArgs) Handles txtPage.KeyDown
        'checks if the spacebar key has been pressed and reacts appropriately.
        'NB. Reference page 513 Deitel and Deitel How to... ISBN 0-13-029363-6
        'for more info on principles used in this section.

        Dim strKey As String

        strKey = pressedKey.KeyCode.ToString
        'the string value of the key that was pressed is stored in "strKey".
        'the integer value of such a key can also be obtained by using pressedKey.KeyCode

        On Error GoTo ErrorHandler
        '**ErrorHandler is used to catch the error if the spacebar is the
        'first key pressed on execution.
        If strKey = "Space" And strWord <> " " Then

            strWord.Trim()
            'the "Trim" function trims the word to be checked from
            'leading and succeeding spaces.

            DoInit(strWord)
            'DoInit will initialize the different components of
            'the word.

            DoCase()

        End If
ErrorHandler:

        Exit Sub

    End Sub
```

Figure 5.4: Testing if the spacebar has been pressed

5.3 Checking words

When the programme reaches the point where a captured word is ready to be checked, before the physical checking algorithm executes, the programme checks if the variable that is used to store the captured word, does indeed contain alphabetic characters. Keyboard keys have an internal character representation and would consequently be stored in the defined variable. This fact means that, if the user pressed the spacebar before any characters have been typed, an empty character would be stored in the variable and influence the checking procedure negatively.

As discussed in chapter four, we decided to employ a partial hash code conversion strategy when it came to storing and checking words. This essentially means that each captured word is transformed into a hash code representation, as are the words in the dictionary. The reason that this strategy was employed was to speed up look-up time and to ensure that each word represented in the dictionary had a unique representation.

If the user pressed the spacebar key before any character had been typed it would mean that the programme would calculate a hash code value for the space character, which in turn will also affect the accuracy of the checking procedure.

This is also true if a captured word is succeeded by a punctuation sign, for example, a full stop. One approach to prevent the programme from assigning hash code values to space characters, words containing space characters and/or punctuation signs was for the programme to ignore space characters and punctuation signs. Figure 5.5 illustrates a fragment of source code that demonstrates the procedure of checking for a space character and for punctuation signs at the end of a word.

Function *InStr* in Figure 5.5 searches the second parameter, which is the base string, for the third argument in the parameter list. If the base string does contain the search string, a zero (0) is assigned to the variable *intResult* and if the base string does not contain the search string, negative one (-1) is returned and assigned to *intResult*.

```
Public Sub DoInit(ByVal strWord As String)
    'strWord references the value of the word to be checked sent from
    'txtPage_KeyDown.

    Dim intResult As Integer
    'intResult stores the return value of 0 or the index
    'where a certain character was found - in this method
    'it's used to identify and remove a fullstop at the end of a word.

    If (strWord.Length <= 0 Or strWord = ".") Then
        Exit Sub
    Else
        If strWord.Length > 1 Then
            intResult = InStr((strWord.Length - 1), strWord, ".")
            'InStr searches the second parameter for a searchstring, the third parameter,
            'and returns the position of where the searchstring is found in the original
            'string, or base string, or returns 0 if the string is not found.
            'The first parameter is used to determine where in the base string the function
            'should start the search, in this case, the second last character in the variable.
        Else
            intResult = InStr((strWord.Length), strWord, ".")
            'if the word typed in has a length of only 1, the search for the fullstop
            'begins at the end of the word.  If 1 is subtracted as above, the value of
            '0 in the first parameter causes the program to terminate because the search
            'for the fullstop cannot begin at position 0.
        End If
    End If
```

Figure 5.5: Checking for space characters and punctuation signs

Consider the else section of the second if: In Figure 5.5 the programme checks if the user has typed only one character before pressing the spacebar button and if it is a full stop.

Figure 5.6 illustrates the source code snippet that removes a full stop from the word if it does contain one, if it does not, the programme continues by sending the captured word to the first segment of the checking procedure.

```
If intResult = 0 Then

    CheckWord(strWord)

Else
    strWord = Left$(strWord, (strWord.Length - 1))
    'Left$ returns a string value represented by the leftmost portion of the
    'first parameter without the value of the second parameter, thus "stripping"
    'the first parameter/string value of the fullstop, which must not be used to
    'calculate the hashcode of the variable.
    CheckWord(strWord)

End If
```

Figure 5.6: Removing a punctuation sign from a word

Once the word does not contain any punctuation marks that could influence its hash code representation upon converting the word to a hash code value, it can be checked against the dictionary for correctness. This is where we employed a strategy similar to that of complete lookup, as suggested and explained in the previous chapter.

We have constructed one electronic main dictionary and two sub-dictionaries [Appendix B]. The main dictionary contains two hundred and thirty three records. Each record consists of seven fields, *HashCode, HashSuggest, Singular, Length, ValidPrefixPlural, Afrikaans and English*. *HashCode* contains the hash code representation of the words. *HashSuggest* contains similarity keys for the words contained in the dictionary from which suggestions for misspelled words can be derived. *Singular* stores the singular form of the South Sotho word whereas *Length* stores the length of each word contained in the *Singular* field. *ValidPrefixPlural* contains the valid prefix(es) for the word in the *Singular* field and *Afrikaans* and *English* fields contains the Afrikaans and English counterparts of the South Sotho word in that record. A sub dictionary, representing all the words in the main dictionary, has been created according to the hash code values of the words in the main dictionary only. Once a word has been converted to a hash code value, it can be checked against the values in the sub dictionary that contain the hash code representation of all the words in the main dictionary. This means that, in actual fact, the algorithm checks the character representation of the words, but faster than would be the case if words had to be compared to each other. The second sub dictionary will be explained later in this chapter.

The programme was designed to assume that a captured word with a length greater or equal to three does consist of a prefix and a stem, or base word. The application splits the captured word into three different parts:

- the whole word

- the prefix
- the base word

In South Sotho, all valid prefixes have a length of two. This means that we could programme the application to extract the first two characters of a word as a prefix through the principle of stripping, assign it to a variable and do a validation to assess if it is a valid prefix and then assess if the extracted prefix is valid for the base word. While the prefix is being stripped off, the remaining portion of the word is also assigned to a variable and checked for validity, as is the whole word.

The section of code displayed in the source code snippet Figure 5.7 displays the use of threading, which enable the programme to better utilize the processing power of the CPU and also enables the programme to do multiple tasks simultaneously.

```
Private Sub CheckWord(ByVal theWord As String)

    If (theWord.Length >= 3) Then
        theWord = theWord.Trim()
        'trims leading whitespace characters without
        'altering the original string.
        strNewWord = New CWordCheck((Mid$(theWord, 1, 2)), _
                                    (Mid$(theWord, 3, theWord.Length - 2)), theWord)
        'the firts argument will be used as the first argument in the
        'parameterlist in the constructor, the second argument as the second, etc.
        'the first argument is the two letter prefix in South Sotho, no prefix
        'larger than two exists, the second argument is the remainder of the word
        'stripped from the prefix and the third argument is the whole word.

        threadCheckPre = New Thread(AddressOf strNewWord.CheckPre)
        threadCheckRemain = New Thread(AddressOf strNewWord.CheckRemain)
        threadCheckWhole = New Thread(AddressOf strNewWord.CheckWhole)
        'create the threads, assign threadstart delegates, in this
        'case the methods in class CWordCheck.

        threadCheckPre.Start()
        threadCheckRemain.Start()
        threadCheckWhole.Start()
        'place each thread in started state.
```

Figure 5.7: Extracting the whole word, prefix and base word as well as multi threading

We utilized a class that contained three private instance variables: one to store the stripped off prefix, one to store the base of the word and the last to store the

whole word. In the third line of code in Figure 5.7, an object of that class is instantiated by the different parts of the word, if the word has a length greater or equal to three. In the three succeeding lines of code, three threads are created and assigned the addresses of three procedures that will each check the different parts of the word for validity and spelling. Each thread is then subsequently placed in its start state and thereafter the procedures are executed simultaneously. If the word has a length smaller than three, the whole word is assigned to the variable that stores the whole word only and validated.

5.3.1 Checking the prefix

Different South Sotho words have different valid prefixes, for example, if the user typed the word “boakere”, the prefix “bo” would be perceived as a valid prefix, because it is indeed a valid prefix in the South Sotho vernacular. However, the word “akere”, has the valid prefix “di”, hence the programme identifies that “bo” is not valid for “akere”.

The procedure that validates the prefixes that have been stripped off the typed word, first validates the prefix against all the valid prefixes contained in a sub-dictionary that only contains valid South Sotho prefixes. If the procedure finds that the stripped off prefix is valid, it checks if this valid prefix is indeed valid for the base word. The algorithm employed in this procedure has been named the *PreCheck* algorithm.

Figure 5.8 illustrates the first part of the prefix-checking algorithm. The private instance variable to which the stripped off prefix was assigned is returned from the class. The variable is used in the prefix-checking procedure.

A variable of type Boolean accepts a value of “true” or “false” from the prefix-checking procedure; a returned value of “true” signifies that the prefix is valid according to the values in the sub-dictionary containing the prefix values and a

returned value of “false” signifies that the prefix was not found and then exits the procedure.

```
strPre = ReturnPrefix()
```

Figure 5.8: Returning the stripped off prefix from the private instance variable from the class

The source code snippet in Figure 5.9 extracts all the records from the prefix sub-dictionary and checks the stripped off prefix for validity. Validation of the prefix value may yield that the prefix is valid according to the returned records contained in the sub-dictionary upon which the value of “true” is returned to the procedure call. The value of “false” will be returned if the prefix is not found in the extracted sub-dictionary records.

```
Try

    daPrefixAdapter.SelectCommand.CommandText = "SELECT * from tblPre"
    DsPrefix.Clear()
    daPrefixAdapter.Fill(DsPrefix, "tblPre")
    drPrefix = DsPrefix.tblPre.FindByPrefix(strPrefix)

    strTemp = drPrefix.Item("Prefix")
    'if the prefix is in tblPrefix in the dictionary
    'strTemp is assigned its value.

    If strPrefix = strTemp Then
        'the value stripped from the word is checked
        'against strTemp to verify if it even exists.
        'MessageBox.Show("Het Pre")
        Return True
    End If
    DsPrefix.Clear()

Catch ex As Exception

    Return False
    'did not find the stripped off prefix to be a valid
    'one.

    Exit Try

End Try
```

Figure 5.9: Part of the prefix checking algorithm

5.3.2 Checking the whole word and the base word

The sub-dictionary that contains only the hash code values, that represent the words in the main dictionary, is utilized as the source of correct words to validate the words typed by the user.

Let us assume, hypothetically, that the user types the word “paate”. The programme strips off the supposed prefix according to the algorithm based on the stripping principle. The result is that the programme assigns “pa” to the prefix variable, “ate” to the remaining word variable and “paate” to the whole word variable. After the algorithm discussed in section 5.3.1 returns a value of “false”, which signifies that the prefix was not a valid prefix in South Sotho, the whole word is validated for correctness.

The source code in the code snippet Figure 5.10 demonstrates the programme assigning the whole word to a variable for validation purposes.

```
strWhole = ReturnWholeWord()
'ReturnWholeWord (modInit) returns the value of
'strWhole from CWordCheck via property Whole
```

Figure 5.10: Assigning the whole word to a variable for validation purposes

A combination of the principles with regard to hashing and complete lookup has been utilized for this particular checking algorithm. Figure 5.11 is a snippet of source code that illustrates the approach we have used. The algorithm employed has been named the *StripHash* algorithm. The first line of code converts the typed word to lowercase. Consider the words “paate” and “Paate”. When converting character strings to hash code values, characters in uppercase will have different hash code values than their lowercase counterparts. Because all the words represented in the main dictionary are represented in lowercase form, and converted to hash code values and stored as such, the occurrence of an uppercase letter in the word will result in a hash code value which, although

the word might still be spelled correctly, will cause the programme to flag the correctly spelled word as incorrect; resulting in a false negative.

```
strFullWord = strFullWord.ToLower
strFullWord = strFullWord.GetHashCode
'the hashcode for the fullword is retrieved.

Try

    daHashCodeAdapter.Fill(DsHashCode, "tblHashcode")
    drHashCode = DsHashCode.tblHashcode.FindByHashCode(strFullWord)
    'the HashCode table is searched for the whole word's hashcode.
    strTemp = drHashCode.Item("HashCode")
    'the found item from the datarow is written to strTemp.

    If strFullWord = strTemp Then
        'MessageBox.Show("Het full")
        Return True
    End If
    DsHashCode.Clear()

Catch err As Exception
    'the hashcode is not found in tblHashCode and causes an error
    'to be raised.

    'MessageBox.Show("Nie full gekry")
    Return False
    Exit Try

End Try
```

Figure 5.11: Part of the whole word and base word checking algorithm

The second line of code converts the word that the user typed to a hash code representation. The sub-dictionary (*tblHashCode*) is accessed and loaded into a data adapter and the programme extracts a record from the data adapter where a pointer indicates the hash code value that matches that of the user's typed word. The procedure returns a value of "true" if the hash code value of the typed word matches one of the records in the sub-dictionary. If the procedure is not able to point to a hash code value in the sub-dictionary that matches the user typed word's hash code value, the procedure returns a value of "false", indicating to the programme that the word is misspelled.

The same algorithm (Figure 5.11) is used to verify both the base word and the whole word, thus employing the principle of code reusability and improving performance.

The code snippet in Figure 5.12 demonstrates the programme's ability to add the words that the application has identified as misspelled to a list box control, thus alerting the user to the fact that a word has been misspelled. In order to accomplish this task, the programme validates the Boolean variables' status "true" or "false", which are returned from the checking procedure.

```
    If boolPreResult = False And boolRemainResult = True Then
        'automatically corrects the word if the prefix is not correct
        'but the remainder of the word is.
        strWhole = strTempPre + strRemain

    If boolPreResult = False And boolRemainResult = False _
    And boolWholeResult = False Then
        lstIdentify.Items.Add(strWhole)
        'neither prefix nor remainder is found in dictionary and
        'added to misspelled-word list.
    End If

Catch err As Exception

    lstIdentify.Items.Add(strWhole)
```

Figure 5.12: Adding misspelled words to a list box

5.3.3 Facilitating a degree of automatic spell correction

One of the main objectives of this study was to enable the programme to have a degree of automatic spell correction. The problem facing a spell checker and corrector programme is that it is difficult to anticipate what word the user actually wanted to type. The user may have wanted to type "akgela", but misspelled the word as "akgeba".

The programme now has two options:

- Change the word automatically to the word with the closest matching spelling that appears highest in the hierarchy of the dictionary.
- Consider the context of the word vis-à-vis the sentence it is contained in and automatically correct it to be context-accurate.

This study did not focus on context sensitive spellchecking and correction, thus the first option discussed above was adopted. This would still not ensure a 100% accurate correction. We decided to re-examine the correlation between a South Sotho word and its prefix, as it was discovered that, in the South Sotho vernacular, words are very often combined with their prefixes in sentences.

This fact meant that, if a word has been typed with a wrong prefix, it could be automatically corrected upon retrieving the word's correct prefix from the main dictionary with 100% accuracy. In case the base word (the remainder of the word when the prefix has been stripped off) is also incorrectly spelled, it will be flagged as incorrect by the programme and upon the user choosing a suggested correction, the system will automatically add the correct prefix, if it was indeed the intention of the user to type the form of the word with its prefix. The system will only realize that the user intended to type the word with its prefix and automatically add the correct prefix, if the user originally typed the misspelled word that the checking-algorithm identified with a valid prefix, even if it was the incorrect prefix for the word. For example, if the user wrongly typed "boakre", "bo" being a valid prefix and "akre" the misspelled word for "akere", the different algorithms will identify "bo" as a valid prefix, but will flag "akre" as a misspelled word. The program will suggest "akere" as the valid spelling for "akre". Upon the user selecting this suggestion from the interface, the programme will automatically correct "boakre" to "diakere", "di" being the correct prefix for "akere".

The user may also type “boakere”, where “akere” has actually been spelled correctly. In such a scenario, the programme will automatically correct the word to “diakere”, without any user interaction. Figure 5.13 contains a snippet of source code that facilitates the automatic correction. The algorithm that facilitates the automatic correction of words is called the *SSAuto* algorithm.

```
            txtPage.Text = Replace(txtPage.Text, (strPre + strRemain), strWhole)
            'automatically changes word in code above.  strPre contains
            'the invalid prefix stripped off from the typed word.
        End If

    End If

End If

If boolPreResult = True And boolRemainResult = False Then
    lstIdentify.Items.Add(strWhole)
End If

If boolPreResult = False And boolRemainResult = True Then
    'automatically corrects the word if the prefix is not correct
    'but the remainder of the word is.
    strWhole = strTempPre + strRemain

    If boolTranslateClicked = True Then
        txtSouthSotho.Text = " "
        txtSouthSotho.Text = strWhole
        'if boolTranslateClicked is true
        'the program must auto-change
        'the misspelled word in txtSouthSoutho
        'and not in txtPage, with what is stored
        'in strWhole.
        cmdTranslate.Focus()
    Else
        txtPage.Text = Replace(txtPage.Text, (strPre + strRemain), strWhole)
        'automatically changes word in code above.  strPre contains
        'the invalid prefix stripped off from the typed word.
```

Figure 5.13: The main section of the automatic spell correction algorithm

5.4 Making suggestions for incorrectly spelled words

The programme required the ability to make suggestions to the user with regard to misspelled words, with a high degree of accuracy. We employed similarity key principles to develop a suggestion algorithm.

First, an algorithm was developed to map each string in the main dictionary into a key value that similarly spelled words would have similar keys and stored in a field in the main dictionary.

We decided to use this method in order to improve system performance, since the misspelled word does not have to be compared to every word in the dictionary directly.

The principle behind similarity keys is that the words are compressed into a key value that will then represent the word. When the programme encounters a character, it calculates a specific key value for the character, in this case a letter, and stores the value together with the other character keys already calculated.

Consider the following key conversion rules based on the letters the programme encounters:

a, e, i, o, u, h, w, y are converted to a zero (0) when encountered.
b, f, p, v → 1
c, g, j, k, q, s, x, z → 2
d, t → 3
l → 4
m, n →5
r → 6

For argument's sake, assume the user types the word "akere", based on the rules mentioned above, the conversion will look like this:

0(a)2(k)0(e)6(r)0(e), that is, 02060

After the initial conversion to the key value, all zeroes are eliminated from the key representation, converting 02060 into 26. Subsequently, all similarity values that

are the same are compressed to only one representation of that letter. The first letter of the original word typed by the user, and indentified as misspelled, is added to front the key value, which means the letter “a” is added to 26, successfully converting the word “akere” to a26.

Assume the user desired to type “akere”, but instead typed “akre”. The checking algorithm flags the word as misspelled and adds it to the list box. The user clicks on the misspelled word and the suggestion algorithm fires, converting the misspelled word to a similarity key for comparison against that found in the dictionary. A similarity key value of a26 is calculated and all words found in the dictionary with matching similarity keys are listed as possible corrections. The user clicks on the desired suggestion and the programme replaces the misspelled word with the chosen suggestion.

Figure 5.14 depicts the code which creates a jagged array and initializes it with the different letters of the alphabet, each row according to the similarity key rules discussed above. The algorithm that has been used to facilitate the retrieval of suggestions is called *SimRetrieve*.

```
strSimilarArray = New String(6)() {}
'allocate the size of 7 to the jagged array (0-based).

strSimilarArray(0) = New String() {"a", "e", "i", "o", "u", "h", "w", "y"}
strSimilarArray(1) = New String() {"b", "f", "p", "v"}
strSimilarArray(2) = New String() {"c", "g", "j", "k", "q", "s", "x", "z"}
strSimilarArray(3) = New String() {"d", "t"}
strSimilarArray(4) = New String() {"l"}
strSimilarArray(5) = New String() {"m", "n"}
strSimilarArray(6) = New String() {"r"}
'initialize the jagged array.
```

Figure 5.14: Creating and initializing the similarity key array

The next figure, Figure 5.15, shows a fragment of source code that is employed to convert each letter of a word into a similarity key.

```
Case 1
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "1"
    End If

Case 2
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "2"
    End If

Case 3
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "3"
    End If

Case 4
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "4"
    End If

Case 5
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "5"
    End If

Case 6
    If strNewString = strSimilarArray(intRowCount)(intColumnCount) Then
        strSimilarKey = strSimilarKey & "6"
    End If
```

Figure 5.15: Converting each letter into a similarity key

The final similarity key is compressed by replacing all repeated values with only one value of that key, eliminating zeros and adding the first letter of the misspelled word to the front of the compressed key.

After the similarity key of the misspelled word has been calculated, the programme can start checking for comparable similarity keys in the main dictionary, retrieve the words with these keys and list them as suggestions with regard to the correct spelling of the word. The programme is constantly aware of the number of records retrieved from the main dictionary. If no suggestions could be retrieved, the user is notified accordingly.

Although South Sotho does not contain highly inflected or compounded words, the occurrence of two words concatenated as one, can not be ignored. This may be due to a user not pressing the spacebar key on the keyboard. For example, the user may have desired to type “akere bofi”, but instead typed “akerebofi”.

The system caters for such a situation through an algorithm derived from the longest string algorithm. The typed word is searched from its front and its back simultaneously for the first string that signifies the longest part of a word, that is, a full word. The algorithm employs a hybrid strategy of both hashing and the principle of similarity keys.

The hash code and similarity keys are thus calculated for strings of characters from the front of the misspelled word as well as the back. The first calculated code of the misspelled word that matches that of a code in the main dictionary is immediately added as a suggestion.

This means that, if the user has typed “akerebofi”, this algorithm will find two full words, “akere” and “bofi”. The algorithm will suggest three corrections, which are added to the suggestion returned from the similarity key algorithm. This implies that the derived longest string matching algorithm will return “akere” and “bofi” as two separate suggestions as well as “akere bofi” as one suggestion.

Even if it is the case that the user accidently concatenated more than two strings, for example, “akerebofibofe”, the algorithm will return “akere”, “bofi” and “bofe” as separate suggestions and “akere bofi bofe” as one suggestion, thus making four suggestions with regard to the typed string.

In Figure 5.16, the source code depicts a snippet of the derived longest string algorithm. This algorithm is called *RetrieFromFB*.

```
For intCounter = 0 To strSingularArray.GetUpperBound(0)
    'reads all the "Singular" records from the dictionary
    'into the array.
    strSingularArray(intCounter) = txtSingular.Text
    'the first index of the array receives the first
    'suggested word.
    'txtSingular is bound to the dictionary's
    'tblComplete's Singular field.  See ISBN
    '0-07-251239-3 for more regarding binding
    'of controls.
    With bmComplete
        If .Position <= .Count Then
            .Position += 1
        Else
            Exit For
        End If
        'checks if there are more records in the dataset
        'to suggest.
    End With
Next
    For intCounter = 0 To strSingularArray.GetUpperBound(0)

        strValue = strSingularArray(intCounter)
        intResult = InStr(1, strWord, strValue)
        '    'searches the second parameter for the
        '    'third parameter
        If intResult <> 0 Then

            If strValue <> strWord Then
                'e.g. user types in "akere" (stored in strWord)
                'and "akere" is found in the dictionary (strValue),
                'there is no use adding it to the error- or suggestion
                'list because no error was actually made although
                'a "false match" was found.
                boolFoundFromFront = True
                'a result was found because intResult
                'has a value.

                intMatchesFound = +1
                'counts the matches found

                If intCounter2 <= 10 Then

                    strReplaceWitharray(intCounter2) = strSingularArray(intCounter)
                    'read in the value so that it can be
                    'used to replace the incorrectly
                    'spelled word in the textbox(es).
                Else
                    Exit For
                    'exits the for if the array is full.
                End If
            Else
                Exit For
```

Figure 5.16: Extracting the longest string in a misspelled word

5.5 Translating South Sotho words

The construction of the main dictionary has facilitated the inclusion of the translation of all the South Sotho words in the dictionary into their Afrikaans and English counterparts. Within each word record of the main dictionary, two fields have been added: one that stores the Afrikaans translation of a South Sotho word and a field that stores the English translation of that same word. A single word in South Sotho may convey a whole idea when translated into another language. For example, the word “akgeha” has the English translation of “to be thrown at, to faint or to go into spasm”.

The translation interface had to check the words the user wanted to translate for misspellings as well, thus all algorithms discussed previously in this chapter were used to validate, suggest corrections and/or automatically correct words.

Through a menu bar, the user can opt to translate a South Sotho word into Afrikaans, English, or both Afrikaans and English. We called the algorithm which retrieves the translations the *DoTranslation* algorithm.

DoTranslation starts off by calculating a hash code value for the word. The algorithm continues to retrieve the record that matches the calculated hash code value. The application reads this record into a data row structure and retrieves the *Afrikaans, English* or both fields of the record, depending on what the user opted to execute via the menu bar controls. The algorithm terminates after the words or phrases extracted, have been displayed to the user, or the user has been informed that the word could not be translated due to the translation not having formed part of the dictionary.

Figure 5.17 shows a snippet of the *DoTranslation* algororithm. The fourth line of code extracts the record in the dictionary where the calculated hash code of the user typed word has matched that of a record. The code in line nine of Figure

5.17 retrieves a field in the record based on what language the user opted to translate into. For Afrikaans, the *Afrikaans* field is retrieved from the record, for English the *English* field and if the user wanted to translate to both Afrikaans and English, both fields are retrieved. The exception handler at the end of the snippet informs the user that no translation could be retrieved for the typed word.

5.6 Performance benchmarks

Software performance is a very important aspect of any software application. Performance benchmarks have been set for this project as well, for both accuracy of the checking- and suggestion algorithms, and performance with regard to the response time of the algorithms. Our target performance benchmarks were as follow:

- achieve at least 75% of accuracy consistently and
- have the algorithms check words, flag misspelled words and make suggestions within 600 milliseconds.

The final product was tested on a machine running the Windows platform with an Intel 1.86 GHz T2350 processor.

5.6.1 System accuracy: Recall measures

The programme's lexical recall, which is the number of valid words in a text corpus that have been recognized as valid by the spell checker in relation to the total number of incorrect words in the text, was deemed an important performance measure.

We employed a native South Sotho speaker to read fifty South Sotho words to a Caucasian focus group. The motivation behind this strategy was to expose the system to as much "abuse" as possible.

```
Dim drTranslate As DataRow
Try
    DsData.Clear()
    daSouthSothoAdapter.SelectCommand.CommandText = "SELECT * FROM tblComplete _
    WHERE HashCode LIKE '" & strWord & "'"
    'retrieve the record(s) where the hashcode
    'matches that of the dictionary.
    daSouthSothoAdapter.Fill(DsData, "tblComplete")

    drTranslate = DsData.tblComplete.FindByHashCode(strWord)

    If boolAfrikaans = True Then
        txtLanguage.Text = CStr(drTranslate.Item("Afrikaans"))
        lblLanguage.Text = "Phetolelo Afrikaans:'"
        '"SS for 'Afrikaans Translation:'"
        txtLanguage2.Text = CStr(drTranslate.Item("English"))
        lblLanguage2.Text = "Phetolelo English:"
        'for in case the user opts to translate
        'to both languages after opting to translatee
        'only to Afrikaans.
    ElseIf boolEnglish = True Then
        'the user wants to translate to English.
        txtLanguage.Text = CStr(drTranslate.Item("English"))
        lblLanguage.Text = "Phetolelo English:"
    ElseIf boolBoth = True Then
        'the user wants to translate to both languages.
        txtLanguage.Text = CStr(drTranslate.Item("Afrikaans"))
        lblLanguage.Text = "Phetolelo Afrikaans:"
        txtLanguage2.Text = CStr(drTranslate.Item("English"))
        lblLanguage2.Text = "Phetolelo English:"
    End If
    lstIdentify.Items.Clear()
    lstSuggestions.Items.Clear()
    'checks which textboxes should be filled with
    'what.

Catch ex As Exception
    lblCountSuggest.Text = "Ha e ya khona ho fetolela lentswe lena."
    MessageBox.Show("Ha e ya khona ho fetolela lentswe lena", _
     "Molaetsa", MessageBoxButtons.OK, MessageBoxIcon.Information)
    'SS for could not translate that word.
End Try

boolAfrikaans = False
boolEnglish = False
boolBoth = False
'resets the values so that when
'the user opts to translate to
'another language, the code above
'will display the correct results.
```

Figure 5.17: Translating a South Sotho word

What better way than to ask a Caucasian, non-South Sotho speaking, focus group to type words read by a native South Sotho speaking individual into *eSpellingPro sa Sesotho sa Leboa*? The pronunciation of the words in the native tongue would make it difficult for a group of people who are not native speakers of the dialect to type the words correctly. Envision a group of Italians typing German words read to them by a German in a German accent.

After the results were analyzed, they yielded that, on average, the focus group spelled forty of the fifty words incorrectly. *eSpellingPro sa Sesotho sa Leboa* confirmed this. Figure 5.18 depicts the results.

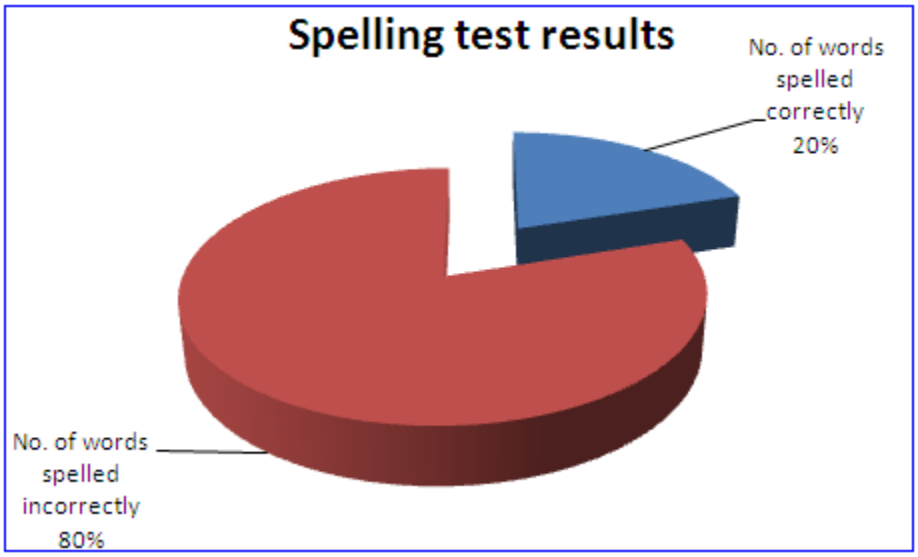


Figure 5.18: Spelling result out of fifty words

The fact that the programme confirmed the results meant it had 100% efficiency and 100% accuracy with regard to lexical recall, that is, identifying misspelled words as misspelled and indentifying correctly spelled words as correctly spelled whilst not flagging correctly spelled words as incorrect nor letting incorrectly words slip by as correct.

5.6.2 System accuracy: Suggestion accuracy

The system exhibited 85% suggestion accuracy. Out of the subset of fifty words chosen to test the system's suggestion accuracy performance, an average of forty words was identified as misspelled. Out of the forty, the words tabulated in table 5.1 were flagged words to which the programme could not suggest the correct word forms or, in some cases, could not make a suggestion vis-à-vis correction at all.

Incorrectly spelled word	Suggestion made by programme	Correct word form for misspelled word
Haboglhoko	Ha, Habo and ha habo	Habohloko
Onifobo	None	Unifomo
Tshatshahetea	Tjotjo and ha	Jajatheha
Meodi	None	Mmeodi
Metso	None	Mmetsa
Methe	None	Mmethe

Table 5.1: Flagged words, suggestions made by the programme and the correct word form

Row two and four of the *Suggestion made by programme* column, serves as proof that the *RetrieveFromFB* algorithm does indeed extract the longest matching string that form actual correctly spelled words from the incorrectly spelled word.

5.6.3 Timed system performance

5.6.3.1 Checking algorithm: Timed performance for correctly spelled words

The first test run was to quantify the time it took for the *StripHash* algorithm to validate words that are correctly spelled as correctly spelled. Ten South Sotho

words where typed, correctly, into the user interface. The words were “paate”, “debita”, “akere”, “akga”, “akgela”, “habohloko”, “tjwebete”, “unifomo”, “ikutlwapelo” and “yare”.

Figure 5.19 portrays the results.

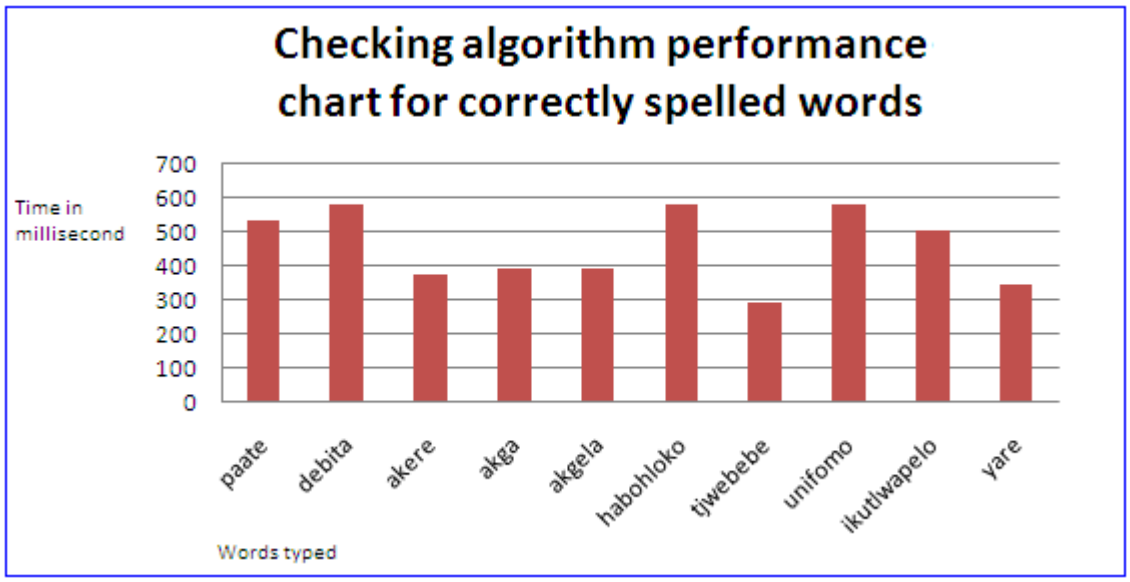


Figure 5.19: Algorithm performance in milliseconds for checking correctly spelled words

As the chart depicts, the checking algorithm performs within the 600 millisecond benchmark when checking correctly spelled words for correctness.

5.6.3.2 Checking algorithm: Timed performance for flagging incorrectly spelled words

In this section, with regard to performance, we quantify the performance of the checking algorithm when it has to flag misspelled words and add the misspellings to a list box. The same ten words were used, but only misspelled on purpose. The algorithm’s performance is illustrated in Figure 5.20.

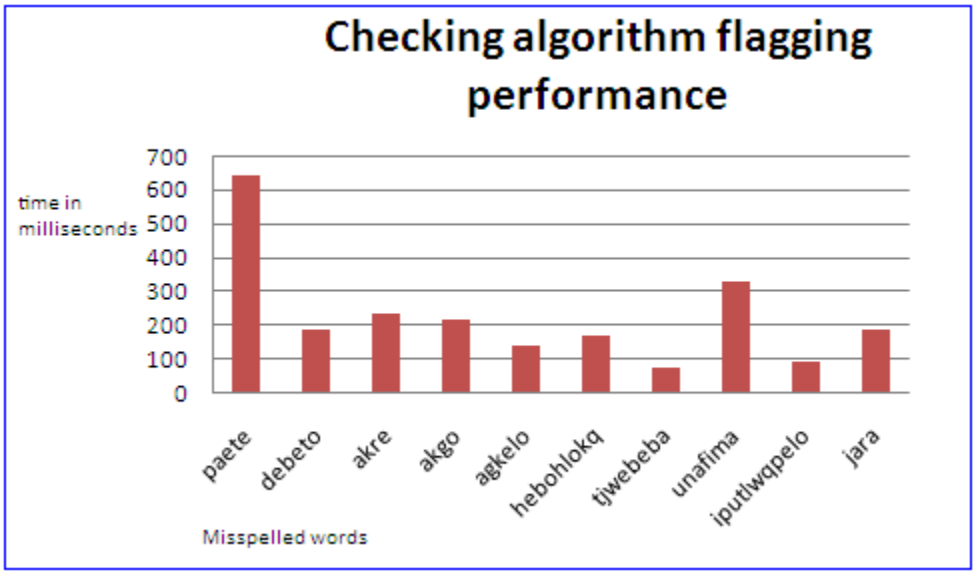


Figure 5.20: Checking algorithm performance with regard to flagging

Except for the misspelled word “paete” (“paate” when correctly spelled), the algorithm performed exceptionally well, with flagging times under the 200 millisecond mark for six of the ten misspelled words. It is in the flagging section of the *StripHash* algorithm that we utilized multithreading, which could explain the exceptional performance of the algorithm.

5.6.3.3 Suggestion algorithm performance: Timed performance for making suggestions for incorrectly spelled words

The next performance test, was in relation to how fast the *SimRetrieve* algorithm could make suggestions of correct word forms for the words flagged as misspelled by the *StripHash* checking algorithm. Again, the benchmark set was a suggestion time of no more than 600 milliseconds.

The accuracy of the suggestion algorithm was discussed in section 5.6.2. Figure 5.21 contains the results of the suggestion reaction times measured. The same misspelled words employed for the test in section 5.6.3.2 were used for the suggestion reaction time test.

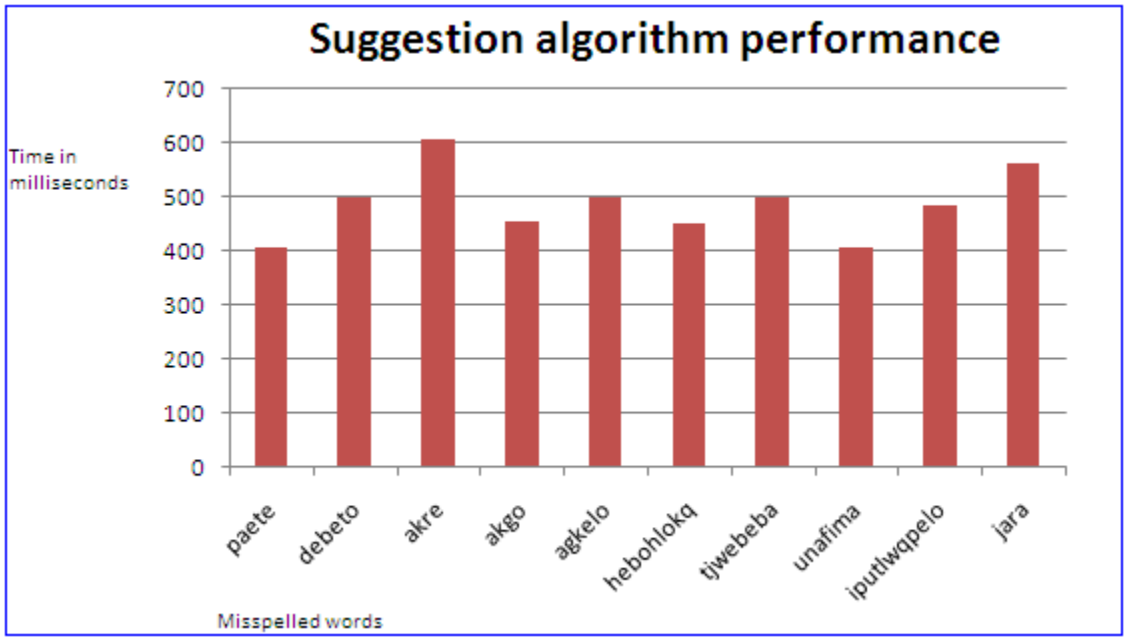


Figure 5.21: Suggestion algorithm performance

The *SimRetrieve* algorithm had satisfactory suggestion times, performing within the defined benchmark.

With regard to all the discussed algorithms, the results depict that the checking and suggesting algorithms yield longer checking- and suggestion times with regard to some words than others. We came to the conclusion that not only the position of the word in the electronic main dictionary had a direct effect on neither the lookup and suggestion times nor so much the length of the word itself, but rather a combination of these two factors. To further clarify this phenomenon, the words found first, in the middle and last in the main dictionary were intentionally spelled wrong. The system actually returned the best lookup- and suggestion time for the word found in the middle of the main dictionary, followed by the word located last in the dictionary. The word located first of the main dictionary had the longest lookup and suggestion time. The test was taken a step further; the words located in the middle of the words found first and in the middle of the main dictionary and in the middle of the words found in the middle and last in the main dictionary respectively, were intentionally spelled incorrectly.

This time, the word located in the middle of the words found in first and in the middle of the main dictionary, returned the worst lookup time of the two.

It can be argued that a larger dictionary will directly affect lookup- and suggestion times. The same can be said for the processing power of the platform employed to run the application on: the platform with superior processing power will return the best system performance. Assuming the application would utilize a larger dictionary, but it is run on a modern processing platform, the user will not notice the effect of the larger dictionary on the system's performance and according to Van Huyssteen [40] this will not adversely affect system performance overall. The test results recorded earlier not only prove that a word located lower (earlier) in the dictionary does not mean that it would have a faster lookup time when compared to a word located higher (later) in the dictionary, but also that it is the combination of a word's position in the main dictionary together with its length that influences lookup and suggestion time. In the employed dictionary design, the main dictionary is sorted according to each word's hash code value in ascending order. Thus, a word starting with the letter "h" would not necessarily be represented in a position after a word starting with the letter "a" in the main dictionary.

5.6.3.4 Translation algorithm performance: Accuracy of translation

The translation algorithm performed at a 100% accuracy rate, checking each word's spelling, automatically correcting it where possible or suggesting corrections and then translating the word into Afrikaans, English or both languages.

5.6.4 Observations

All the algorithms developed, yielded more than satisfactory results, both with regard to accuracy and performance. Only one exception existed within the

selected corpus of words to be tested, where the checking algorithm performed just outside the set benchmark (see Figure 5.20).

The rest of the results from the algorithm performance tests had an acceptable success rate. This provided the feasibility of using this application on a normal desktop computer for spellchecking and correcting of South Sotho words, whilst boasting the ability of a degree of automatic spell correction and translation of South Sotho words into its Afrikaans and English counterparts.

It is recommended that the system is installed on a computer running Windows XP as the operating system for the reason that the application was developed, tested and fine tuned on this platform.

With regard to performance, if the user wishes to obtain similar algorithm performances as in the performance tests in this chapter, it is suggested that an Intel 1.86 GHz T2350, or newer, processor and a system with 512 MB RAM or more be used.

The final chapter will discuss problems that still exist, possible solutions for these problems and some conclusions arising from this work.

Chapter 6

6. CONCLUSION AND FUTURE WORK

The development of the spellchecking and correcting programme, *eSpellingPro sa Sesotho sa Leboa*, was a challenging and yet an exceptional learning experience. Not only did the development of this system force the researcher to delve deeper into the functionality of and explore the abilities of the programming language that has been utilized; which we found very enriching in its own right, but it also promoted creative problem solving, innovative thinking and an investigation into existing software programmes related to this study.

6.1 Algorithm implementation, performance and multi threading

A variety of algorithms were developed for this study. The algorithms that were employed in the final system where selected based upon their performance and accuracy aspects or trade offs. The programme was created for output performance statistics upon the completed execution of each respective algorithm. After the statistics were extracted, the lines of code responsible for generating the statistics were removed from the functioning code.

To further enhance algorithm performance, we decided to make use of an object oriented programming approach and implemented the algorithms via a multi threaded procedural execution strategy, rather than making use of the traditional sequential programming methodology and flow of the procedures. The strategy utilized not only enhanced the algorithms' performance, but also exploited the CPU's processing power optimally. Code reusability was also kept in mind with regard to the development of the algorithms as well as their implementation, which can make them reusable in another system. Although these algorithms and procedures are reused in the system itself, they were not compiled separately for redistribution purposes.

Algorithms were loaded into separate threads of execution and the threads were put into their start states when the algorithms were needed. This meant that several procedures could be executed simultaneously. The algorithms employed performed above the set benchmarks, with regard to both accuracy and performance. An accuracy benchmark of 75% was set, which meant that three out of four suggestions made by the system, would be correct. A study conducted by Van Huyssteen and Van Zaanen yielded precision measures (how accurately the programme assigns non-flags) of 29.17% to 72.02% and suggestion accuracy of 77% to 87% [42]. The algorithms developed yielded an accuracy figure of 85% on a subset of words.

There was one case where the checking algorithm under-performed. In that particular case the algorithm validated an incorrectly spelled word, flagged it as incorrect and added it to a list box after 641 milliseconds; 41 milliseconds higher than the set benchmark of 600 milliseconds. It was concluded that the position of the word in the dictionary lead to the lower than expected performance upon the first execution of the programme, although sub-performance of the checking algorithm did not occur again. The best checking, flagging and listing time measured was 78 milliseconds.

When the checking algorithm was tested for validating correctly spelled words, the worst validation time achieved was 578 milliseconds and the best validation time 291 milliseconds. The suggestion algorithms also yielded very good results. The longest suggestion time within the set benchmark measured was 563 milliseconds, whereas the best time with regard to making a suggestion for a misspelled word measured 406 milliseconds. Automatic word correction was made at an average time of 388 milliseconds, which the researcher deemed to be acceptable.

The system utilized multi-threading in order to allow several algorithms to execute simultaneously. The algorithms that validate the correctness of the prefix, the base word and the whole word are executed in parallel, returning the results to be utilized by the programme at different stages. Our initial idea was to also put the *RetrieveFromFB*, the algorithm that searches the typed word for the longest correctly spelled strings, into

its own processing thread, but after some consideration, this was not done. It was decided to allow *SimRetrieve*, which is the algorithm that calculates a similarity key value for an incorrectly spelled word, to attempt to retrieve suggestions first. The *RetrieveFromFB* algorithm fires when the *SimRetrieve* algorithm is not able to supply the programme with suggestions and attempts to extract a suggestion from the misspelled word.

The implementation of the threading methodology proved to be beneficial to system performance, allowing the algorithms to comply with the set performance standards and even performed well enough to flag incorrectly spelled words in less than 100 millisecond times in many cases.

6.2 Satisfying the research hypothesis

As stated in chapter one, the objective of this study has been to prove that:

A spell checker and corrector application for the South Sotho language can be developed and run successfully and reliably on a personal desktop computing system, whilst:

i. *Providing a satisfying ratio between spell-error identification and automatic correction of misspelled words. This entails the application having the ability to automatically correct misspelled words where possible or to identify incorrectly spelled words if it can not be automatically corrected.*
ii. *Employing the ability to suggest correctly spelled words for misspelled-words.*
iii. *Performing reliably, that is, it should not flag correct words as incorrect or observe incorrect words as correct.*
iv. *Having the ability to translate the meaning of a South Sotho word into Afrikaans and English.*
v. *Perform within set benchmarks with regards to checking-, flagging- and suggestion -performance and -accuracy.*

For the hypothesis to be satisfied, the following criteria had to be met:

i. *The application must flag incorrectly spelled words as incorrect as well as not flagging correct words as incorrect (True positive- (Tp), True negative- (Tn), False positive- (Fp) and False negative (Fn) identification). A true positive is a valid word recognized by the spell checker and corrector, resulting in a correct non-flag. A true negative means a word that is incorrectly spelled and recognized by the system, which results in a correct flag, or "good flag". The meaning of a false positive is as follows: invalid words are not recognized by the application, resulting in incorrect non-flags, whereas a false negative means the programme flagged a validly spelled word as incorrectly spelled, that is, an incorrect flag.*
ii. *The success rate of error identification (flagging) is no less that 90% and employ a degree of automatic word-correction.*
iii. *The application must produce results consistently, that is, 75% of the time with no less than 75% suggestion accuracy.*
iv. *The application must perform flagging and suggestion within a time frame of 600 milliseconds.*
v. *The application can supply the Afrikaans and English words having the same meaning as the identified South Sotho word.*

The various tests run and the results obtained from these tests, presented in chapter five, serve as proof that the study hypothesis was satisfied.

i. In section 5.6.1 of the previous chapter, the test results showed that the system demonstrated 100% accuracy when considering correctly spelled words and lexical recall. This meant that the programme did not perceive incorrectly spelled words as correct (false positive) and also did not flag correctly spelled words as incorrect (false negative). All misspelled words were flagged (true negative) and all correctly spelled words were perceived as correctly spelled, resulting in a true positive identification.

ii. Section 5.6.2 discussed the testing methodology that was employed and the results that the tests yielded. The system exhibited 85% suggestion accuracy, which meant that the algorithm responsible for making suggestions for incorrectly spelled words, performed at 10% above the set standard.

iii. Figures 5.19 through 5.21 depict how the various algorithms performed. Only one case existed where the checking and flagging algorithm did not perform up to expectation, performing 41 milliseconds above the benchmark. In all other cases, all algorithms performed consistently, reliably and with satisfactory response times, well below the set benchmark.

iv. The application incorporated a degree of automatic spell correction of misspelled words with 100% accuracy, performing automatic corrections at an average of 388 milliseconds.

v. Finally, to totally satisfy the study hypothesis completely, the programme has a 100% success rate vis-à-vis the translation of South Sotho words into Afrikaans, English or both languages.

Lexical recall has been calculated using a formula suggested by Van Huyssteen [41]:

$$Rc = \frac{Tp}{Tp + Fn}$$

where *Rc* represents “recall correct”, *Tp* represents “true positives” and *Fn* represents “false negatives” [41]. The total number of valid words in the corpus, that are recognized by the programme, were used in relation to the total number of correct words in the text, that is, the sum of all the false negatives and true positives.

Error recall (Er) has been calculated by using the following formula:

$$Er = \frac{Tn}{Tn + Fp}$$

Tn (True negatives) represents the number of invalid words in the corpus that have been correctly flagged by the application in relation to the total number of incorrect words in the text, meaning the sum of all true negatives and false positives. Note that "false positives" relate to words that have been spelled incorrectly, but not identified by the programme.

Precision accuracy, that is, how accurate the spell checker and corrector is, have been calculated using the formula:

$$Pc = \frac{Tp}{Tp + Fp}$$

Pc represents *precision recall*. It was calculated by dividing all true positives found in the text by the sum of all true positives and false positives found in the text, which represents the total number of words that were not flagged as incorrect, irrespective of whether a word was spelled correctly or not.

Finally, overall performance (*Op*) was quantified using the following formula:

$$Op = \frac{Tp + Tn}{Tp + Fn + Tn + Fp}$$

This test was done to calculate exactly how accurate *eSpellingPro sa SeSotho sa Leboa* actually performed. Within the limited scope of this study, the developed programme performed with an overall performance of 100%. By combining the derived

principles of stripping, calculating hash code values for typed words to simplify lookup as well as the concept derived from the complete lookup strategy, the programme performed with the mentioned 100% accuracy when it came to validating words, identifying misspelled words and flagging them as such, whilst not flagging correctly spelled words as incorrect.

Finally, the suggestion measures were quantified using the formula:

$$Sa = \frac{Ns \times 100}{Tn}$$

Sa represents “suggestion accuracy” and *Ns* (“No suggestions”) represents the true negatives that the programme listed as incorrectly spelled, but could not find the correct suggestions or any suggestions at all. *Tn* represents “true negatives” again: the words that have been perceived by the programme as misspelled and correctly flagged as misspelled words. The suggestion accuracy figure for the spell checker and corrector was 85%.

The system performed consistently with regard to both accuracy and time. As the figures in 5.6.3 in chapter five suggest, with the exception of one case, the system’s performance benefited from the implementation of a multi threaded execution methodology that was utilized throughout different sections of the programme. Combining the approaches of calculating similarity keys and complete lookup also proved beneficial with regard to suggestion accuracy. The algorithm that was developed to search a misspelled word for the longest matching string, to a validly spelled word also served as a successful, accurate and helpful extra resource to the primary suggestion algorithm.

6.3 Suggestions for further work

The test results that have been presented in chapter five and discussed further in this chapter have satisfied the entire criteria set for the successful completion of this study. The system is not 100% perfect and that there is room for improvement. Listed below are some inadequacies of this work that could form the basis of further work in the future:

i. The current system only recognises the full stop sign as a valid punctuation mark. As previously explained, the inclusion of a punctuation mark in a captured word will affect the calculation of a hash code value, which will result in the identification of a false negative.
ii. When the user becomes aware of a mistake made before the system's algorithms fire and attempt to rectify it, the user has to erase the entire string from the back of the word up to the point where the correction is intended. The user cannot simply move to the intended point of correction with the mouse or arrow keys and make the correction as such (see section 5.2).
iii. Instead of using the algorithm that searches a misspelled word for the longest valid matching string, an autonomous agent with the ability to learn could be employed for faster and more accurate suggestion results and a wider scope of automatic correction.
iv. The system does not currently have the ability to translate more that one South Sotho word or phrase at a time. The functionality to translate more that one South Sotho word or phrase was not included in the final solution. When the translation of more than one South Sotho word is considered, we would have had to validate the context in which the words appear. That falls outside the scope of this study.
v. The current programme cannot define a South Sotho word as denoted in a dictionary, due to the fact that different forms of the same word are used in different contexts, which, as stated, falls outside the scope of this study.

6.4 Discussion

The research presented in this study demonstrated that the system developed, the programming and execution methodology utilized as well as the algorithms that have been developed, implemented and tested, have been capable of identifying misspelled words, automatically correct these incorrectly spelled words or have suggested correctly spelled words for them, consistently, accurately, reliably and within set time benchmarks.

Even though there are aspects of this system that can be optimized or have not been perfected, which may benefit from future research, the system provided proof that it is capable of performing successfully and as intended. The algorithms developed have been implemented with great success as was the use of the underlying technologies such as multi threading.

This system has been developed to pave the way for spellchecking and correcting applications for native African languages. Although the system has limited scope, there is no doubt that steps in the right direction have been taken to empower the new age of computer application users, especially with regard to spellchecking and correcting software applications. The developed application is open to further research and could be implemented as a learning or business tool which could serve and benefit both these areas equally.

This chapter concludes the work that has been done.

"What you get by achieving your goals is not as important as what you become by achieving your goals."
Zig Ziglar

"Great works are performed, not by strength, but by perseverance."
Samuel Jackson

REFERENCES

1. 19h00 Afrikaans News Bulletin, 12-02-08

2. Aduriz, I., et al (1993). *A Morphological Analysis-based Method or Spelling Correction. In Proceedings*, Sixth Conference of the European Chapter of the Association for Computational Linguistics, Utrecht, The Netherlands, 463 – 464.

3. Carter, D.M. (1989). *Lexical Acquisition in the Core Language Engine*. In: Proceedings of the 4th European Chapter ACL Conference, ACL. Manchester. Great Britain. April 1989. pp. 133 – 144.

4. Chanal, P. (2007). *Belangrikheid van goeie kommunikasie*. **Die Volksblad**, 13 January 2007.

5. Communications of the ACM, December 1980, Volume 23 (12).

6. Corin, R.E. (1980). *SPELL: A Spelling Checker and Correction Program*. Documentation for Stanford Spelling Checker, Private Communication.

7. Dagobahn, E. (2007). *Why are spelling skills deteriorating?* [Online]. Available from: http://www.lucasforums.com

8. Damerau, F.J. (1964). *A Technique for Computer Detection and Correction of Spelling Errors.* Communications of the Association for Computing Machinery, 7(3): 171 – 176.

9. Du Plessis, J.A., Gildenhuys, J.G., Moila, J.J. (1978). *Bukantswe Ya Maleme-Pedi.*

10. Dwyer, D. (1997). *Webbook of African Languages.* [Online]. Available from: http://www.panafril10n.org

11. Finkler, W. and G. Neumann (1988). *MORPHIX – A Fast Realization of a Classification-Based Approach to Morphology.* Bericht Nr. 40. XTRA. University of Saarlandes, 11pp.

12. Gazdar, G. and Mellish, C. (1989). *Natural Language Processing in PROLOG. An Introduction to Computational Linguistics.* Addison-Wesley Publishing Company, Reading, MA.

13. Golding, A.R. (1995). *A Bayesian hybrid method for context-sensitive spelling correction.* Mitsubishi Electric Research Labs.

14. Hajič, J. and Drózd, J. (1989). *Spelling-checking for Highly Inflective Languages.* Research Institute of Computing Machinery. 3, 1, 1 – 8.

15. Hoffmeister, B. (2006). *Is grammar deteriorating in the teaching profession?* [Online]. Available from: http://www.helium.com

16. http://www.dictionary.com, accessed on the 11th of June 2007.

17. http://www.living-e.de, accessed on the 18th of February, 2008.

18. http://www.lookwayup.com, accessed on the 18th of February 2008.

19. http://www.thefreedictionary.com, accessed on the 18th of February 2008.

20. Knuth, D.E. (1973). *The Art of Computer Programming, Volume 3: Sorting and Searching.*

21. Kukich, K. (1992). *Techniques for Automatically Correcting Words in Text.* ACM Computing Surveys, 24: 377 – 439.

22. Lamprecht, L.B. (2007). *Jonges se blitstaal het 'n lelike draai.* **Die Volksblad,** 29 August 2007.

23. Oflazer, K. (1994). *Error-tolerant Finite-state Recognition.* Computational Linguistics, Volume 22 (1).

24. Paggio, P., Underwood, N.L. (1997). *Validating the TEMAA LE evaluation methodology: a case study on Danish spelling.* Natural Language Engineering. 1(1): 1-18.

25. Peterson, L.J. (1980). *Computer Programs for Detecting and Correcting Spelling Errors.* ACM Volume 23 (12676 – 687).

26. Peterson, L.J. (1980). *Computer Programs for Detecting and Correcting Spelling Errors.* University of Texas at Austin, Volume 23 (12).

27. Peterson, L.J. (1980). *Computer Programs for Spelling Correction.* Lecture Notes in Computer Science Volume 96, Springer Verlag.

28. Pollock, J.J. and Zamora, A. (1984). *Automatic Spelling Correction In Scientific And Scholarly Text.* Volume 27 (4).

29. Pretorius, C. (2005). *SMS-taal rampspoedig vir kinders se taal-, spelvermoë.* **Die Volksblad**, 16 April 2005.

30. Prinsloo, D.J. and De Schryver, G.M. (2004). *Spellcheckers for South African Languages Part 1: The Status Quo and Options for Improvement*. Dept. of African Languages, University of Pretoria, S.Afr.J.Afr.Lang.

31. Randall, P. (2005). *Texting responsible for deteriorating levels of spelling and grammar.* [Online]. Available from: http://mocoblog.com

32. Rosenbaum, W.S. and Hillard, J.J. (1975*). Multifont OCR postprocessing system.* IBM J. Res. And Develop. 19, 4, 398 – 421.

33. Ruch, P. (2000). *Information Retrieval and Spelling Correction: an Inquiry into Lexical Disambiguation.* Fe4deral Institute of Technology, University Hospital of Geneva. CoNLL-2000 (ACL-SGNLL) Proceeding, pages 111 – 115.

34. Smith-Atakan S., Polak J., Krashen S. (2006). *Human Computer Interaction.*

35. Steffler, D.J., Varnhage, C.K., Fiala, P. (1986). *The Computer as a teacher of Czech.* In: Proceedings of SOF-SEM '86, Volume 11. UVT UJEP Brno. JCMF. Liptovsky Jan. Nizke Tatry. 1986. pp. 187 – 190.

36. Steffler, D.J., Varnhagen, C.K., Friese, Christine K., Treiman R. (1998). *Journal of Educational Psychology, Volume 90.* [Online]. Available from: http://www.questia.com

37. Tanaka, E. and Kasai, T. (1972). Correcting *method of garbled languages using ordered key letters.* Electronics and Comm. In Japan. 55, 6, 127 – 133.

38. TESOL quarterly report: There's *More to Children's s Spelling Than the Errors They Make.* Volume 22 ((1), pp. 141-146).

39. Turba, T.N. (1979). *Checking for Spelling and Typographical Errors in Computer-Based Text.* Language Systems, Roseville Development. Center. Volume 14 (12, 92 – 109), University of Alberta, Rebecca Treiman.

40. Van Huyssteen, G.B. (2006) and Van Zaanen, M.M. *A spellchecker for Afrikaans, based on morphological analysis.* Potchefstroom University for Christian Higher Education, Potchefstroom, South Africa and Van Zaanen, M.M., University of Amsterdam, Amsterdam, The Netherlands.

41. Van Huyssteen, GB and Janke, U. (2005). *Developing Spelling Checkers for South African Languages.* Unpublished paper presented at "International Conference of the African Languages Association of Southern Africa", University of Johannesburg, JOHANNESBURG, South Africa.

42. Veronis, J. (1988*). Morphosyntactic Correction in Natural Language Interfaces.* In Proceedings, 13th International Conference on Computational Linguistics, 708 – 713, Wayne State University.

43. Zamora, A. (1976). *Control of Spelling Errors in Large Databases: The Information Age in Perspective.* Proceedings of the ASIS Annual Meeting Volume 15, (364 – 367).

Appendix A

A. PROGRAM DESIGN

This part of the dissertation outlines the most important aspects of *eSpellingPro sa Sesotho sa Leboa*. Programme components and functions are listed and every aspect is accompanied by a diagram which outlines the procedural flow of that particular aspect. The top most section of the procedural flow diagrams indicate the process or the form from which processing has been passed.

A.1 Main user interface

The user interface utilized a straight forward, easy to use design. The form in Figure A.1 is loaded by default and is the main work area of the programme. It also acts as a link to the form from which the translation of South Sotho words is done.

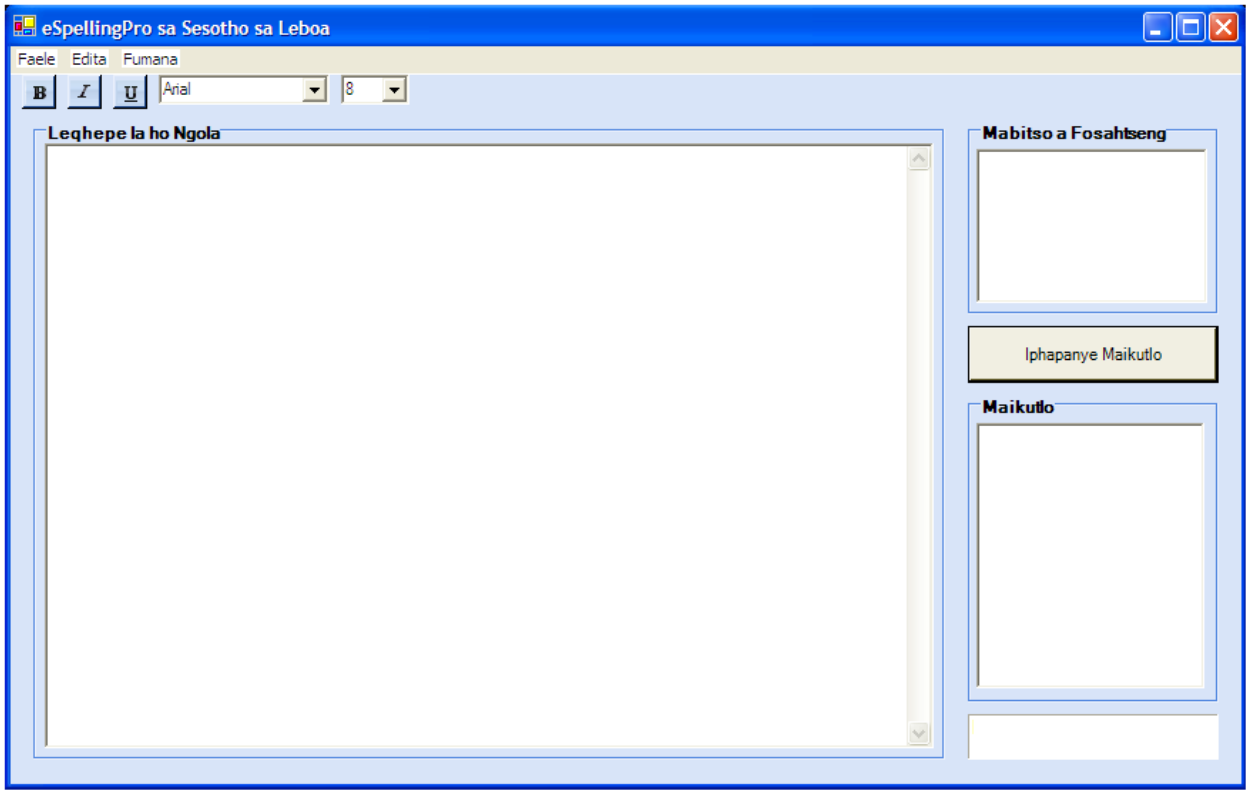


Figure A1: The main form

A.1.1 Procedural flow diagram: Main interface

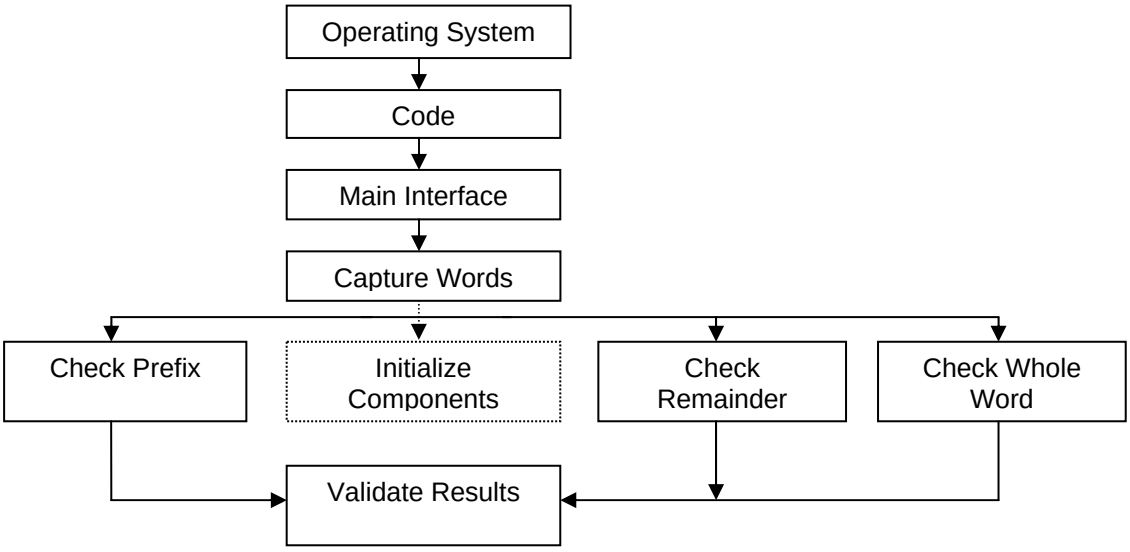


Figure A.2: Main interface procedural flow diagram

A.1.2 Main menu mnuMain

The main menu, mnuMain, contains various drop down menu controls that contain items usually associated with that control. Table A.1 depicts each menu control and its associated items.

Main menu item	Sub-menu item	
Faela (File)	Print	
	Kwala (Exit)	
Edita (Edit)	Ketha Tsohle (Select All)	
Fumana (Translate)	Fetolela Ho (Translate To)	
		Afrikaans
		English
		Afrikaans le English
	Khutlela ho Leqhepe la Ngola (Back to Main Typing Page)	

Table A.1: Menu- and sub-menu controls of mnuMain

Sub-menu item *Print* allows the user to print the typed text. *Exit* exits the application whereas *Select All* selects the whole text in the main text area. The *Translate To* menu item formats the form to allow the user to interface with the system in order to translate words from South Sotho to Afrikaans, English or both, each represented as a menu item. The *Back to Main Typing Page* item returns the main form to its original position when clicked upon.

A.1.3 Toolbar tlbFormat

tlbFormat is a menu consisting of icons which represent familiar text formatting tasks, such as to bold text, to underline it, change the font style and so forth. Table A.2 describes the toolbar items and their function.

Toolbar item	Function
cmdBold	Changes text to boldface.
cmdItalic	Changes the text to italic.
cmdUnderline	Underlines text.
cboFont	Changes the font type.
cboSize	Changes the font size.

Table A.2: Toolbar items and their function

A1.4 Other controls on frmSS_SpellCheck

Table A.3 contains the control name and its function for all other controls on the main form.

Control	Function
txtPage	The main typing area from where the programme captures, validates and flags words,
lstIdentify	Misspelled words which have been flagged by the system are added to this control. These words provide suggestions when clicked upon and are removed when a suggestion is chosen.
cmdIgnore	Informs the program to ignore a misspelled word in the text and removes the flagged word from lstIdentify.
lstSuggest	Lists suggestions. When a suggestion is clicked upon, the misspelled word in the text corpus is changed to the chosen one and all other suggestions are removed from lstSuggest and the flagged word is removed from lstIdentify.
lblCountSuggest	The number of suggestions found are displayed here and the user is informed via lstCountSuggest if no suggestions could be found.

Table A.3: Controls of the main form

A.2 Translation user interface

The translation form is employed to allow the user to utilize the programme to translate South Sotho words into the two other listed languages, Afrikaans and English. Figure A.3 displays the translation form.

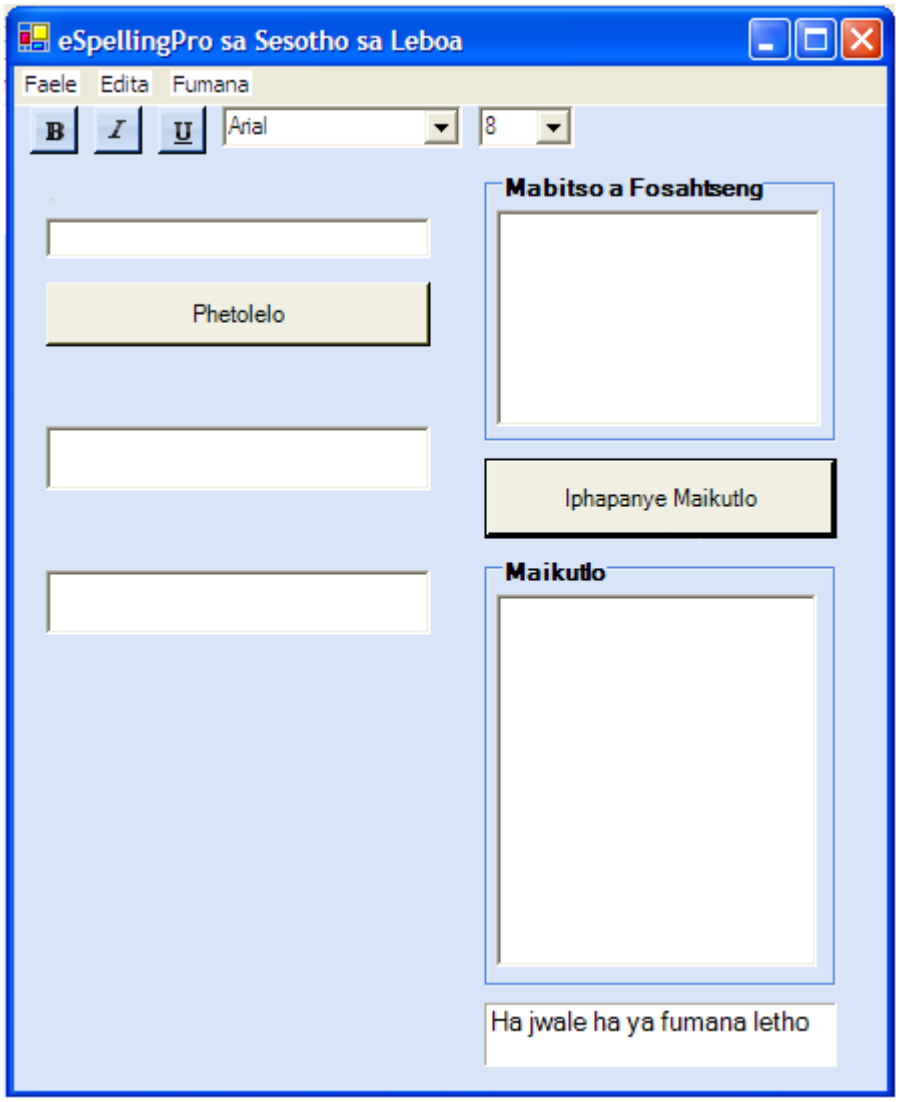


Figure A.3: The translation interface

The functionality of the menu controls, the controls contained in the toolbar, lstIndentify, cmdIgnore, lstSuggest and lblCountSuggest have already been discussed. Table A.4 tabulates the additional controls and their functionality.

Control	Function
txtSouthSotho	The textbox where the user enters the South Sotho word to be translated.
cmdTranslate	Invokes the translation algorithm.
txtLanguage	The word translated into Afrikaans or English is displayed here, based on what language the user opted to translate to.
txtLanguage2	When the user opts to translate to both Afrikaans and English, the English translation is displayed here.

Table A.4: Additional controls of the translation interface

A.2.1 Procedural flow diagram: Translation interface

Figure A.4 depicts the procedural flow of the translation interface in its entirety.

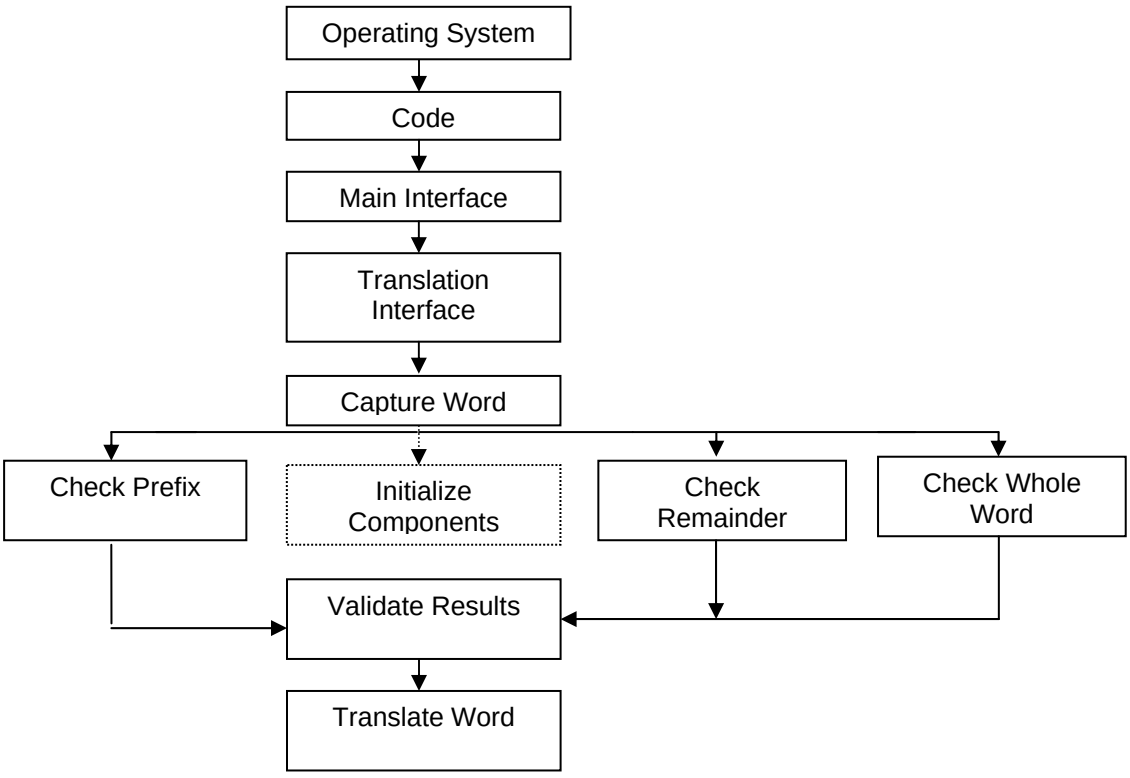


Figure A.4: Translation interface procedural flow diagram

A.3 Procedural flow diagram: Checking algorithm

Figure A.5 illustrates the full procedural flow of the checking algorithm.

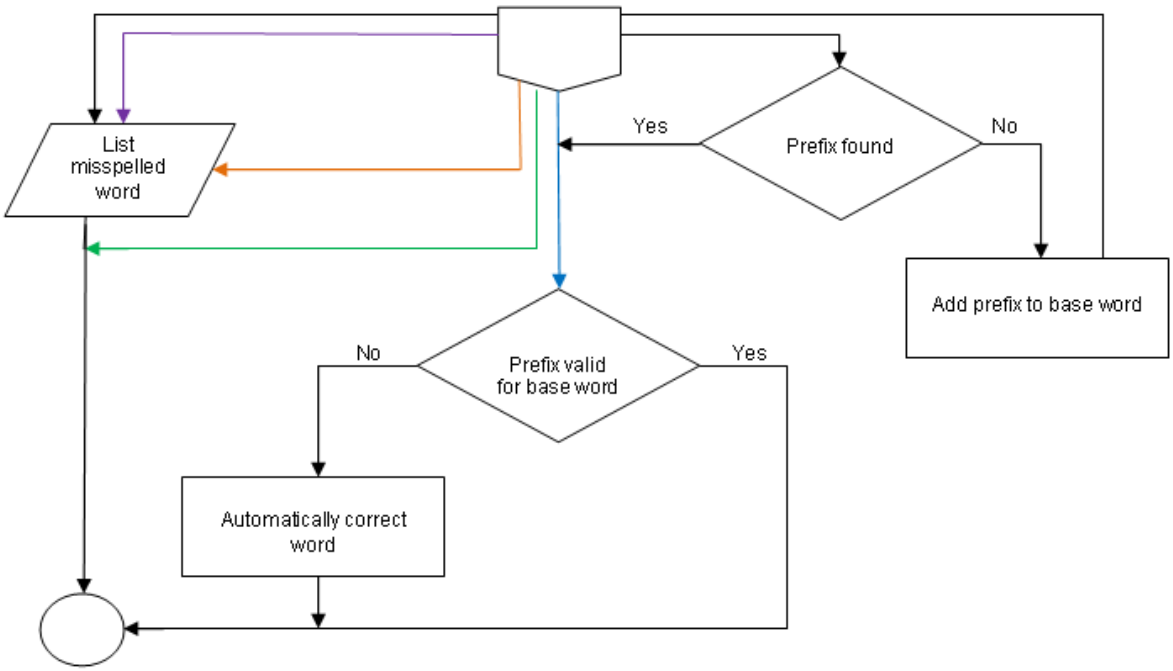
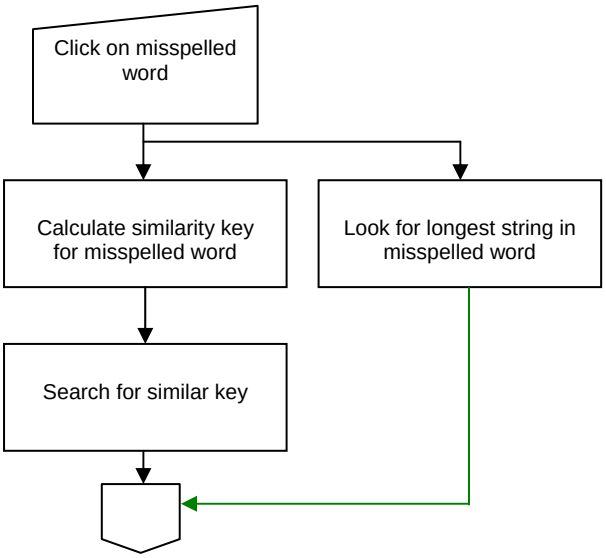


Figure A.5: Procedural flow chart for checking algorithm

A.4 Procedural flow diagram: Suggestion algorithm

In Figure A.6, the suggestion algorithm's procedural flow diagram has been depicted.



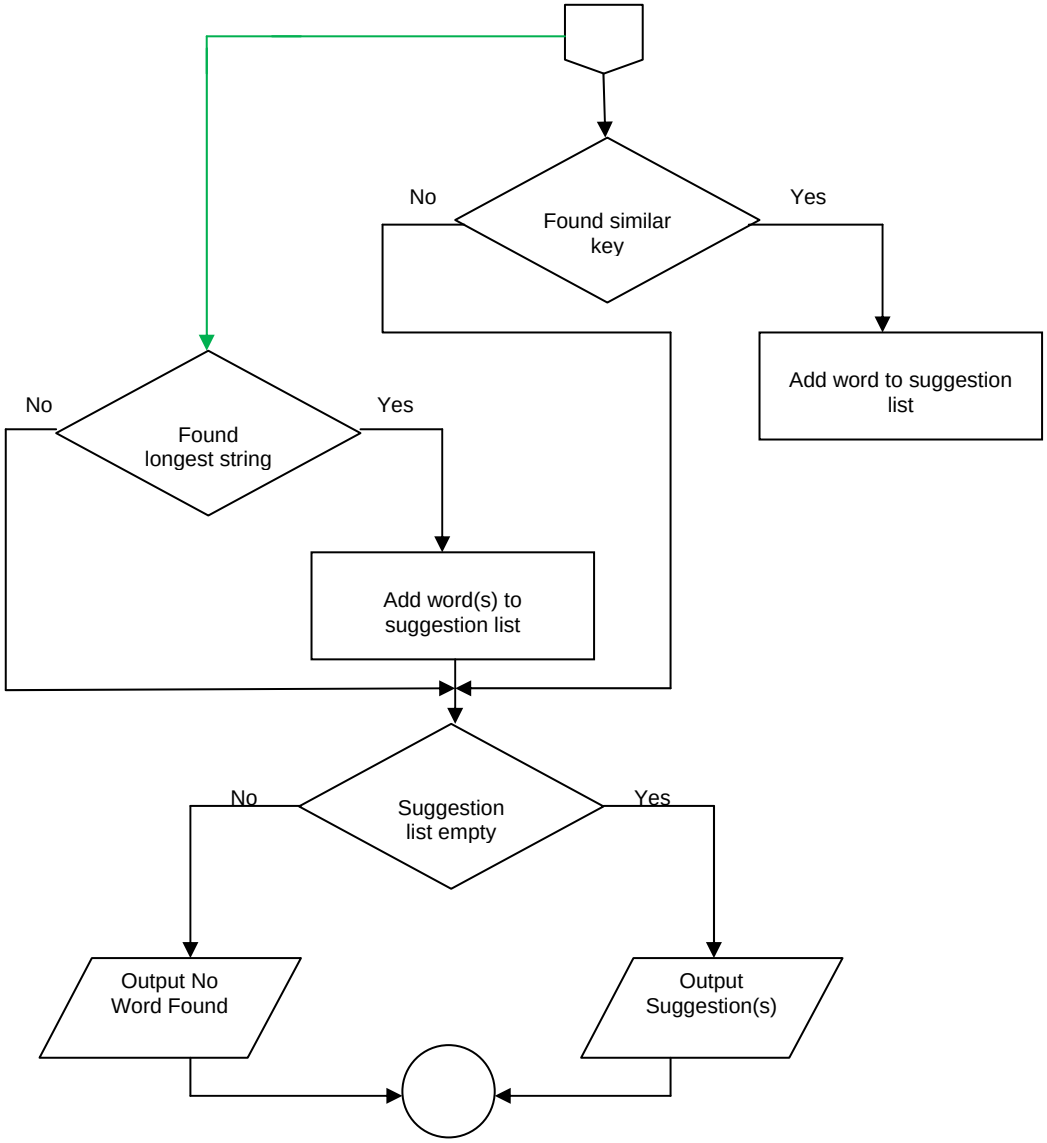


Figure A.6: Procedural flow chart for suggestion algorithm

A.5 Procedural flow diagram: Translation algorithm

The procedural flow diagram for translation algorithm:

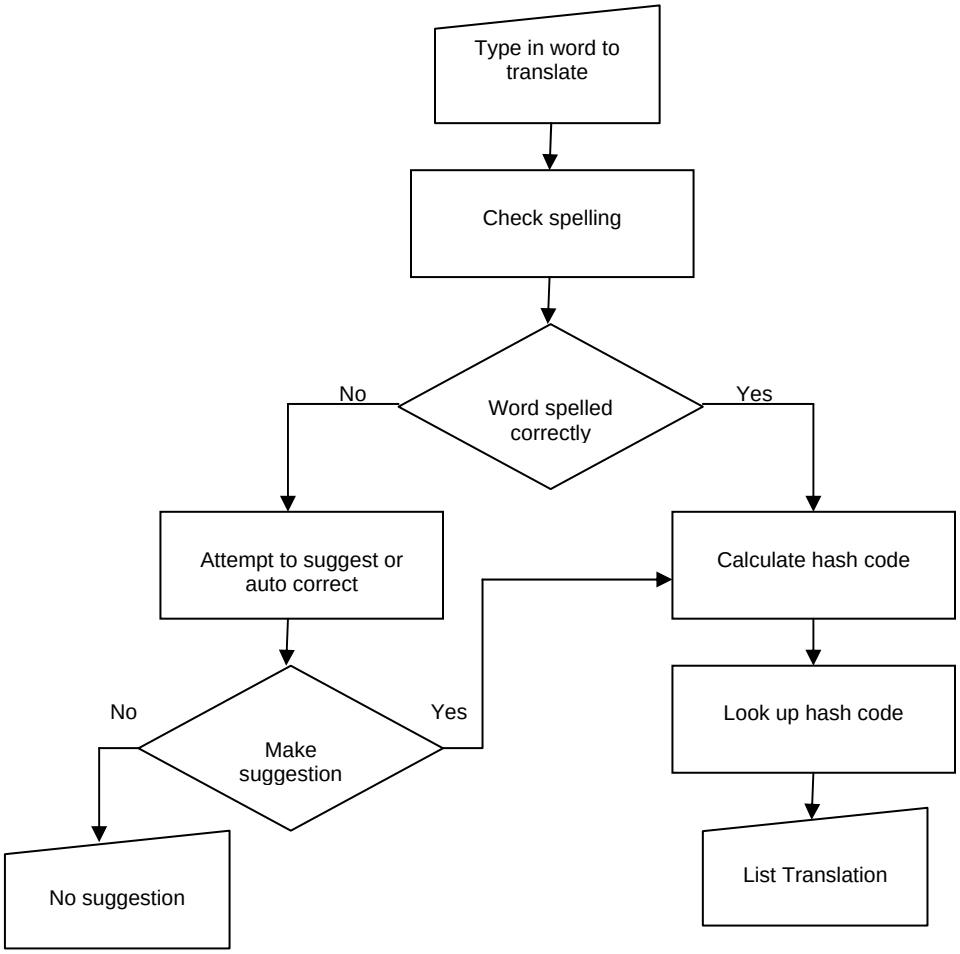


Figure A.7: Procedural flow chart for translation algorithm

Figure A.7 concludes this section.

Appendix B

B. DICTIONARY DESIGN

In this section, we would like to present a short discussion on the dictionary design and its implementation. A custom electronic dictionary was manually created and is unique to this study by using *Bukantswe Ya Maleme-Pedi*, which is a South Sotho/English dictionary. Approximately nine words for each letter of the alphabet were randomly extracted from the paper based dictionary and utilized in the electronic dictionary, where possible. We state the "where possible" for the reason that there are some letters of the alphabet that, in South Sotho, do not appear at the beginning of any words, for example, there are no words that begin with the letter "c" in South Sotho and there are only two words that start with the letter "x", that is, "Xhosa" and "Xrei". A total of two-hundred-and-thirty-three words exist in the lexicon.

One main dictionary and two sub dictionaries were created and the dictionaries are represented as tables in Microsoft Access. Each dictionary serves a different purpose with regard to both validation and suggestion. Different words correspond to different records within each table, which include a set of fields for each matching word. The tables were also linked to enforce the referential integrity of the database to ensure that all updates were made in all the necessary records and fields as well as to ease the process where records had to be extracted for checking and suggestion purposes or to be presented to the user.

B.1 The main dictionary

This dictionary contains two-hundred-and-thirty-three records, each with seven fields. The fields are *HashCode, HashSuggest, Singular, Length, ValidPrefixPlural, Afrikaans* and *English*. Field *HashCode* contains the hash code value that was calculated for

each singular word, in lowercase, contained in the dictionary, which is stored in the *Singular* field of each record. It was initially thought to store different hash codes in the *HashSuggest* field of the dictionary. This would have entailed that we would have had to manually search for words that are similarly spelled as other words and store their hash codes next to each other with some kind of delimiter to be used for suggestion purposes. This approach would have been tedious and would have left little room for the dictionary to be expanded, especially if a user dictionary was also employed in future. For these reasons, the *HashSuggest* field stores calculated similarity keys for each word based on the rules discussed in section 5.4. These values are then used by the suggestion algorithm to suggest correctly spelled words to the user. The *Length* field stores the length of each word contained in the *Singular* field. *ValidPrefixPlural* stores the valid prefix for each word of the *Singular* field. Fields *Afrikaans* and *English* stores the Afrikaans and English translation of a particular word. The main dictionary is alphabetically sorted according to the *Singular* field.

Figure B.1 provides a partial look of the main dictionary.

tblComplete : Table

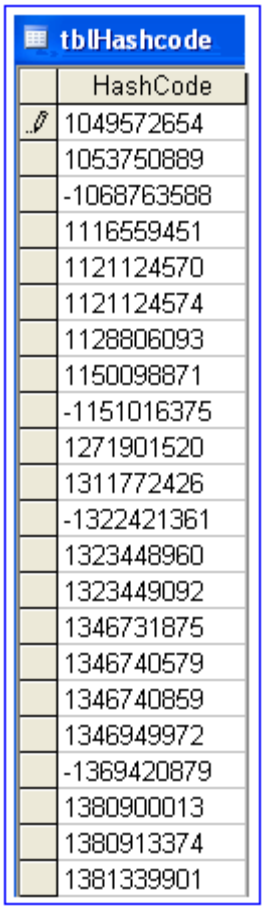
HashCode	HashSuggest	Singular	Length	ValidPrefixPlural	Afrikaans	English
-1477729889	a263	akaretsa	8	none	opsom of saamvat	to conclude or wrap up
170261693	a26	akere	5	di	akker in terme van grootte van 'n stuk	an acre with regards to the size of a e.g. field
2087567625	a2	akga	4	none	gooi, hande waai of afkeur	to throw, wave arms or disapprove
807617149	a2	akgaseakgo	10	mo	skoppelmaai	to thorw the throwable or a swing that children play o
1323449092	a2	akgeha	6	none	gegooi word, flou word of stuipe kry	to be thrown at, to faint or to go into spasm
1323448960	a24	akgela	6	none	gooi vir	throw to
-1873090353	a245	akgelana	8	none	gooi vir mekaar	throw to each other
2087567617	a2	akgi	4	se	'n dans van Sotho mans of skoppelma	a dance of Sotho men or ?
724911680	a21	akhaefe	7	di	argief	archive
211138441	B53	bantu	5	none	bantoe	bantu
2087730447	b1	bofa	4	le	bind, boei, vasmaak of pakdier	tie up or cuff with handcuffs, manacles or chains or a
2087730443	b1	bofe	4	le	grastou of koringtou	grass- or wheat rope
2087730439	b1	bofi	4	mo	persoon wat vasbind	the person that ties up
1500748850	b12	bofjwa	6	se	gevangene	prisoner
-2015078720	b14	bofolla	7	none	losmaak, aflaai of ontlaai	to untie, loosen, unload or offload
1500766701	b1	bofuwa	6	mo	persoon wat vasgebind is	the person that is tied up
2087730113	b	boha	4	none	kyk na, bewonder of staar	to look at, admire or stare at
1500394956	b	boheha	6	none	mooi	beautiful
175617325	b4	bohla	5	none	wind opbreek, verminder, bulk	to break wind, lessen or the low of cattle
175617321	b4	bohle	5	none	almal	everyone
445437216	d132	daebitsi	8	none	suikersiekte	to have diabetes
1116559451	d	daehwa	6	none	word gekleur	being colored by e.g. paint
-1808645694	d5	daemane	7	di	diamant	diamond
163985875	d2	daese	5	di	dobbelsteen	dice related to gambling
-1798882242	d532	damutsa	7	none	skiet	shoot
-2079321677	d53	danemaete	9	di	dinamiet	dynamite
2087377941	d3	data	4	di	data	data
1121124570	d13	debita	6	none	debiteer	to debit
1121124574	d13	debite	6	none	debiet	debit
164133880	d3	deita	5	none	dateer	to date
193405315	e1	eba	3	le	duif	dove
193405319	e1	ebe	3	none	en dan	and then
164995842	e14	ebile	5	none	en toe, verder	further more
164996601	e12	ebisa	5	none	duiselig maak, verblind, verbyster	to make dizzy, to blind or to cover something up
165001984	e14	ebola	5	none	afskil, skil	to peel, peel
-701458295	e14	ebolela	7	none	afskil vir	to peel for
1150098871	e14	ebolwa	6	le	skil	peel
193405517	e3	edi	3	mo	grens of perk	border or limit
-478179742	e354	edimola	7	none	gaap	yawn
193407976	e	e-e	3	none	nee	no
1346949972	f1	fapoha	6	none	wegdraai, uitdraai of uitgaan	to turn away, evade or to go out

Figure B.1: Partial view of the main dictionary

B.2 Sub dictionaries

B.2.1 Hash code sub dictionary

The hash code sub dictionary was created to improve the algorithms' lexical recall time. It also contains two-hundred-and-thirty-three records, but only one field, named *HashCode*. This field contains each of the calculated hash code values for each word in the main dictionary. When a word is typed and subsequently captured by the system, the checking algorithm calculates a hash code value for the typed word and compares it to the hash code values stored in the *HashCode* field. If a hash code value is found, it means that the typed word was correctly spelled, and vice versa. Figure B.2 depicts a partial view of the hash code sub dictionary.


tblHashcode

	HashCode
✎	1049572654
	1053750889
	-1068763588
	1116559451
	1121124570
	1121124574
	1128806093
	1150098871
	-1151016375
	1271901520
	1311772426
	-1322421361
	1323448960
	1323449092
	1346731875
	1346740579
	1346740859
	1346949972
	-1369420879
	1380900013
	1380913374
	1381339901

Figure B.2: Partial view of the hash code sub dictionary

B.2.2 Prefix sub dictionary

This sub dictionary was created to serve the application in terms of referencing all valid prefixes when the prefix of a word has been stripped-off. Figure B.3 represents a full view of the prefix sub dictionary. The dictionary contains one field, *Prefix*, wherein all valid South Sotho prefixes are stored. The prefix validation algorithm checks the stripped-off prefix for validity by referencing each of the records in this dictionary. If the prefix that has been stripped-off, matches one of the records in the sub dictionary, it is flagged as being correct, but, because it does not mean the particular prefix is correct for the word it was concatenated to, the stripped-off prefix is then also checked against the *ValidPrefixPlural* field of the main dictionary after the base word validation algorithm has returned its applicable value. If the valid stripped-off prefix does not match the prefix found in the *ValidPrefixPlural* field, the automatic correction algorithm executes. If the two prefixes match, it is passed by the prefix validation algorithm as being correct. When the prefix is not found in the sub dictionary, the programme only checks the whole word for a misspelling.

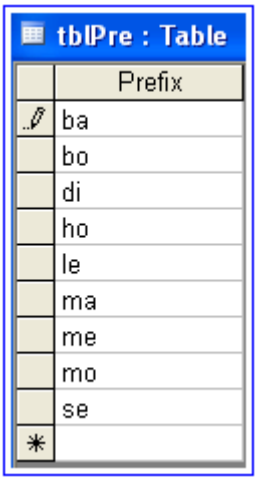


Figure B.3: The prefix sub dictionary

Section B.2.2 concludes this segment of the work.

Appendix C

C. THE QUESTIONNAIRE

Presented in this section is an *English* version of the questionnaire to which the student focus group was subjected to. It was decided to translate the questionnaire into English based on the assumption that not all the readers of the work would be South Sotho speaking. Spelling mistakes were made on purpose throughout the questionnaire for the reasons stated in section 3.1 and were not compromised during the translation. Refer to chapter 3, section 3.1 for a more detailed description of why exactly the questionnaire was utilized.

C.1 The physical layout of the questionnaire

The questionnaire's layout was as follows:

Questionnaire

A study to determine what students think the reasons are for the deterioration of the correct spelling of words among South Sotho speaking groups.

Number of subjects (size of focus group): ***40 people***

Further Information: ***The subjects on whom this particular study is based upon are second year students of the Central University of Technology, FS' Welkom campus. The focus group will be instructed to complete the***

questions posed in as most detail as possible, incorporating their own views on each question. Most questions will be open ended questions without leading the answer and students will remain anonymous at this stage to further encourage honest views and opinions.
The construction of this questionnaire was done by following the principles discussed in "Human-Computer Interaction, Serengul Smith-Atakan, 2006".

Questions

Please answer the questions below. Please write neatly and legibly. Elaborate on the open ended questions to the widest extent possible, but keep your answers question-spesific. Please also answer the whole qeustion and not just a part of it.

1. Is it your opinion that the abilety to spell correctly among South Africans is deteriorating?

_____.

2. What about South Sotho specificaly; do you think that the correct spelling of South Sotho words among South Sotho speaking people is also deteriorating?

_____.

3. If you answered yes to either of the above mentioned questions, please answer the following: In terms of current and evolving technologies, what do you percieve as contrebuting factors for this deterioration of "spelling skills"?

_____

_____

_____

_____

4. What are your views on our current school-system regarding the promotion of proper spelling-skills among scholers? Feel free to list positive and negetive views.

5. Is it your opinion that mixing our langauges has an effect on the ability to spell correctly? Please answer “Yes” or “No” followed by a short elaboration of your answer.

6. Regarding txt read by the everyday South African e.g. magasines and other literature; what effect do you perceive this media-form to have on spelling capebilities of the reader?

7. Do you feel that other media forms e.g. tellevision has a deteriorating effect on the spelling skills of our citizens? What about the music people listen to? Please elaborate your answer.

8. What are your views regarding the reading of books (e.g. Stephen King novels etc.) and its effect on spelling? Does the youth read more or less?

9. How much do you use a spelling-checker/corrector like Microsoft Word? In your wiev, what are the benefits of such a program? Please also list the negative aspects you feel these programs have.

10. What extra features would you like to be abel to use in a spelling-checker/corrector? What are the current shortcomings of these programs according to you?

11. Can you think of other factors that contrebute/can contrebute to deterioration of spelling skills among South Africans as well as the rest of the world?

12. Please indicate on a scale of 1 to 5 (1 being the least/lowest and 5 being the most or highest) which of the following you feel contribute to spelling deterioration e.g. if you feel rap music is a strong contender for increasing

spelling mistakes, a rating of 4 or a 5 would be appropriate, if you feel it does not really influence spelling mistakes a rating of 1 or 2 would be sufficient.

Reason / Medium	Rating (1 to 5): (1 – Not a contributing factor at all) (2 – Cannot be held totally accountable) (3 – Has accountability of approximately 50%) (4 – Seen as a factor that can be seen as contributing extensively regarding spelling deterioration) (5 – Can be held very accountable, has big influence on the way spelling deteriorates)
Television Programs	
Radio Lingo (words or phrases used by DJs)	
The fact that people read less	
Youth magazines	
The use of cell phones regarding text messaging	
The use of e-mail	
The lack of proper protocols in place to enhance and advance spelling capability on school level	
Music like rap, kwaito, hip-hop	
The availability of spelling correcting software and its use	
The mixing of languages	

13. Before reading this specific question, did you notise any spelling mistakes among these questions and text? Start your answer with either “Yes” or “No”. If u have, how many and at wich questions or where in the text? Please list the incorrect spelled words, according to you, as well as thier correct versions. If you did not, what do you think the reasons are? Please be honest, the survey is completly anonymous and your honesty will be greatly appreciated and assist me to identify certain tendencies.