Spelling Correction for Filipino using Double Metaphone and Levenshtein Distance

Joanna Marie Ompoc and Mikee A. Villanueva

June 26, 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Description

To use a word also means to spell it correctly. According to (Gredler, 2002), spelling is important in order to fluently read. This is because spelling and reading follows the same mental representation of a word. Learning how to spell means learning information about the word's print, speech sounds, and meaning(Moats, 2005). Content writing involves both spell checking and proofreading, and in order for writers to reduce their time in writing, they often forget these important features. Spelling errors on published works may lead to readers not being able to understand the content of your work and they may lose their interest in reading your work because of all the distractions within the content itself. Sometimes, poor spellers tend to only use words that they can spell in their writings. This leads to the lack of verbal power and their work may only focus on the spelling rather than the content itself(Moats, 2005).

Spelling errors are sometimes caused by hitting the wrong key on the keyboard, cognitive errors like phonetics, and OCR errors which is generally caused by visual similarity. Phonetic errors poses a greater problem in solving spelling errors because these misspellings often have more than a single insertion, deletion or substitution error (Kukich, 1992). Phonetic spell checkers often use Soundex, originally used for coding and indexing family names, for generating spelling suggestions. Through soundex, any words can be compressed to just a 4-digit code - the first letter of the word follow followed by 3-digit numbers. Spelling correctors often have very large dictionary data thus ways to compress these data must be done so that the dictionary won't take up too much space (Zobel & Dart, 1995). Soundex, however, often produces inefficient spelling suggestions because of its limited code combinations (Zobel & Dart, 1996). Another method used in phonetic spell checkers is called Metaphone, an improvement to Soundex, which encodes words according to how it sounds. Also, another extension to Metaphone is the Double Metaphone which produces both primary and secondary codes whereas metaphone only produces one(Philips, 2000).

Various techniques are used in ranking suggestions like Wagner and Fischer's sting-to-string correction problem (1974), Boyer-Moore's fast string matching (1977), Knuth's fast pattern matching (1977), Sellers's evolutionary distances (1980), and Levenshtein's self-correcting codes (Blank, 2012). The string-to-string correction problem is used to figure out the difference between two strings by using the minimum cost sequence of operations in order to transform one string into the other(Wagner & Fischer, 1974). Boyer-Moore's fast string matching algorithm finds the location of a string on another string(Boyer & Moore, 1977) while Knuth's fast pattern matching algorithm determines how many times a string appears within a string with a run-time that depends on the length of each string(Knuth, Morris, & Pratt, 1977). The evolutionary distance method determines the pattern of similarities between two strings(Sellers, 1980). Lastly, Levenstein's self-correcting codes that calculates the smallest possible insertion, deletion, and substitution to transform one string to the other(Levenshtein, 1966).

A Filipino spellchecker by (Cheng, Alberto, Chan, & Querol, 2007) called SpellCheF which is composed of modules such as dictionary-lookup, n-gram analysis, Soundex, and character distance measurements. SpellChef used two methods for generating suggestions - Tri-gram analysis and Soundex. These suggestions were ranked by using character distance method which uses a Pythagorean-type metric. The results showed that more suggestions are generated by using the Soundex approach. However, the tri-gram approach produces more concise suggestions faster than Soundex.

Filipino may almost be the same as Tagalog but it contains "western" letters like f, j, c, x and z. ("Tagalog Lang", 2002). Filipino can be referred as a subset of Tagalog because it includes contributions of languages other than Tagalog. Also, foreigners favors "Tagalog" than "Filipino", and some overseas Filipinos tend to follow what word the foreigners considers more (Gucasan, 2017). The phonetic rules for the Double Metaphone will be based from the Komisyon sa Wikang Filipino or KWF's "Ang Ortograpiya ng Wikang Pambansa".

This study aims to create a Filipino spelling corrector that will focus on the phonetic structure of each words. Two databases will be utilized for the Filipino dictionary and for the common words dictionary. The Filipino dictionary will contain over 21,000 words which will be retrieved from [https://github.com/jmalonzo/tl-wordlist], a GNU licensed dictionary originally used for Aspell.

Furthermore, the study will be using Double Metaphone to produce spelling suggestions, and Levenshtein Distance to rank the generated suggestions. Then, the results from the aforementioned methods will be memoized to avoid recursions. Each words in the dictionary will be transformed into their Double Metaphone encodings and they will be grouped according to their encodings. The input will also be converted into the encoding and will be matched in the common words dictionary first before the Filipino dictionary. The edit distance of each of the suggestion from the input will be determined using the Levenshtein Distance algorithm and only a maximum of 10 suggestions will be shown to the user.

1.2 Statement of the Problem

Most of word processing programs provides spelling checker and corrector to insure the correctness of their work. Without using any methods we mentioned in Section 1.1, the traditional way to detect error and correct those errors is to compare the given text in every words in the dictionary and on the condition that the given text is not found in the dictionary, the given word will then be considered as a misspelled word. Assuming that the given text is misspelled, the system still need to retrieve the correct word of the text. However, deciding which word is correct can be a challenge for the system since the misspelled word may have several possible correct words. Furthermore, having such a big dictionary can cause a slower running time to the system resulting to a slower program, and aiming suggestions accuracy is also a challenge.

1.3 Research Objectives

1.3.1 General Objectives

To provide a Spelling Corrector for Filipino using the Double Metaphone and Levenshtein Distance.

1.3.2 Specific Objectives

1. To create phonetic rules using the manual entitled "The Filipino Orthography" from the KWF or Komisyon sa Wikang Filipino.

2. To provide spelling suggestion using Double Metaphone

3. To provide ranking for the suggestions using the Levenshtein Distance.

4. To utilize the memoization technique for faster retrieval of data from the *commonwords* database.

5. to compare the performance using the metaphone and levenshtein method and using the levenstein method.

6. To evaluate the performance of the system on its ability to check spelling and provides suggestion for misspelled words.

1.4 Scope and Limitation of the Study

Document files, that are encoded in .txt format, in Filipino language will be used as test subjects in this study. The study only limits the use of text files instead of documents that contain images, tables and graphs. Furthermore, this study focuses on generating spelling suggestions rather than the detection of misspellings within the document.

1.5 Significance of the Study

Spelling checker for Filipino language is significantly important especially for Filipino, since it can help further development and enrichment of native languages such as Hiligaynon, Chabacano, Cebuano, and other different language in the Philippines. Furthermore, spelling corrector also plays a significant role in learning a language, thus having this study may help other individuals in learning the Filipino language. This study also creates the metaphone rules for Filipino Language which are beneficial for researchers with a similar study.

Chapter 2

Review of the Related Literature

The topics that are included in this chapter helps to gain information that are relevant and similar to the present research. The topics below includes discussions on Spell Checker, Phonetic Matching, Levenshtein Distance, and Memoization.

2.1 Spell Checker

2.1.1 Filipino Spell Checking and Correction

SpellCheF is a Filipino spell checker that uses dictionary-lookup, n-gram analysis, Soundex, and character distance measurement for the detection and correction of misspelled words (Cheng et al., 2007). The spell checker is composed of the lexicon builder module, n-gram analysis module, detection module, and correction module. The extraction of each words from the source materials is located at the lexicon builder module. These words will be categorized according to the rules from KWF rulebooks, the 2001 Revision of the Alphabet and Guidelines in Spelling the Filipino Language, and the Gabay sa Editing sa Wikang Filipino. In the n-gram analysis, the system executes a non-positional tri-gram analysis where three n-gram look-up tables will be generated. These tables contains the KWF lexicon, Gabay lexicon, and the Common Words lexicon. The detection module's work is to decide whether a word is correctly spelled or misspelled. Since the system is a plug-in to OpenOffice Writer, this module will be called on this software. Then, it is in the corrector module where suggestions will be generated. This module utilizes both Soundex code and Tri-gram Analysis. Finally, after generating the list of correct spelling suggestions, these words will be ranked by using character distance method which is used to arrange and trim the suggestions. To rank the suggestions from the aforementioned method, the Pythagorean-type metric was used to determine the edit distance between the misspelled word and a suggestion.

According to their result, the Corrector module achieved a 94% accuracy rate while their Detector module achieved an error rate of 7% which means that some correct words were flagged as misspelled. Reasons for this error are small lexicon, low frequency of the tri-gram, and two-letter words that are not included in the lexicon. Also, the length of the misspelled word and the suggestions greatly affect the rank of the suggestions because when the suggested word and the misspelled word has the same length, then that suggestion will appear on top of the list.

Furthermore, according to their results on the suggestion module, even though Soundex produced more suggestions than the tri-gram approach, produced more efficient suggestions than Soundex. This is because Soundex always accepts the first letter of every word as correct.

2.1.2 Afrispel: An isiZulu Spell Checker

Another spell checker for an African language uses an algorithm where it will take in an input text which will be tokenized and then it will search for these words in the exception list. The words are then individually searched in the exception list that is collected from the corpus. The exception list contains known correctly spelled words. This will help minimize the number of words to be taken through the error detection algorithm. The words that are not on the exception list, referred to this study as the non-word errors, will be taken to the error detection algorithm. For detecting non-word errors, the researchers used the combination of the character-trigrams or character-four-grams and the probability of having each character-trigram or four-gram in a particular training dataset. The researchers also used a hash table for the exception list because it is fast and it minimizes the number of comparisons for the lookup (Ndaba, n.d.).

The system's spell checker has a high accuracy rate when detecting words that are not in the training sets. The three corpora that are used in the n-gold cross validation also achieved a high-performance rate. The Ukwebalana corpus, which achieved the highest result, had an accuracy rate of 85% with a threshold of 0.003, Prof. Langa corpus had an accuracy rate of 67% at 0.003, and finally, the news items corpus achieved an accuracy of 76% with 0.003 thresholds. However, testing the system with an outdated corpus can lead to a poor performance where Ukwebalana had 50% accuracy rate. This shows that a spell checker's performance highly depends on its corpus. Furthermore, the spellchecker performed better with trigrams compared with four-grams because it is most likely to find trigrams of a word than its four-grams.

Although their study doesn't include suggestions, according to the researchers, the error detection of the system is their first priority. Also, the spell checker flags infrequent words as wrong spelling. The researchers suggest an improvement in the data-driven statistical language model with a theory-driven linguistic model when building a spell checker. With this, the spell checker would be able to check if an infrequent word follows the language rules, and this may improve the system's performance. Lastly, a spell checker should also be able to detect unlikely combinations of words.

2.1.3 GNU Aspell

An Open source spell checker called GNU Aspell can be used as a library or a standalone spell checker (Atkinson, 2016). It is specially designed to replace Ispell - a program that can correct spellings and typographical errors in any files (*Ispell*, 2006). Aspell does the best word suggestion replacements for misspelled word for the English language, it can easily check documents in UTF-8 without using a special dictionary, it also includes support for multiple dictionaries at once, and it can handle personal dictionaries when multiple Aspell process is open at once (Atkinson, 2016). Aspell's process, as described by (Atkinson, n.d.), is listed below:

1. The misspelled words will be converted to the correct word with almost the same sound.

2. It lists all words that sounds alike with only one or two edit distance from the source word. When it is set to only look for words with only one edit distance, it will compare it to all possible words that sound alike, and each words will be checked in the dictionary. But when two edit distance is to be found, it will scan the whole dictionary, and it will pick up all suggestions. The spell checker is fast because it limits only until two edit distance.

3. It will then find common misspelled words that have the same correct spelling replacement and store it in the same suggestion list as if it was a correct spelling.

4. The suggestions will be ranked and the suggestion with the lowest score will be returned.

5. Lastly, the correctly spelled word will be replaced to the misspelled

word.

2.2 Phonetic Matching

2.2.1 SoundEx

A study by (Stanier, 1990) used statistical tests into a couple of data sets which contains 411,716 surnames to establish the accuracy of Soundex coding. According to Alan, only a third of the matches that will be found are correct but a quarter of correct results won't be discovered. The results of his study shows that only 33% of the Soundex results are correct and the 25% genuine matches that were not located was the major problem. According to Stanier, the shorter the code, the higher the number of incorrect matches will be found because given surname pairs like *Cholmondely* and *Chumley*, both surnames will be variants of the same soundex code but *Askey* and *Haskey*, and *Lloyd* and *Floyd* won't even be a match.

According to (Christian, 1998), the following principles are the three main principles used in creating Soundex rules:

(a) Sometimes, the features of some surname contains too many variable and it is hard to be coded efficiently. Vowels are important in the English dialect variation, and it is hard to create an easy and efficient vowel-coding scheme that includes obvious variants.

However, single or double consonants may be highly variable but they are not very significant.

(b) There are letters that sounds alike but treated individually. Variants like 'd' and 't' sounds alike and can be interchanged because they are identified as "dental stops" for the English language.

(c) Letters that are used differently, and letters that are used because they sound alike are coded in one group. The letter 'g' can be used for both 'g' and 'j', and the letter 'c' can be used for 's', 'k', and 'ch'. These two letters with their corresponding sound alike are coded into a single large group.

However, there are two fundamental principles of spelling that are ignored in Soundex. These principles listed below are very important in the English language:

(a) There are some combination of letters that represent a single sound. 'dg' (sometimes spelt as 'j') is one sound and not a combination of the single letters 'd' and 'g'. Also, the combinations 'ch' and 'tch' have a similar sound but both are coded by Soundex differently. Thus, it makes 'Bachelor' and 'Batchelor' as matches.

(b) Consonants that are clustered together don't produce one sound, and an easier spelling is reflected on this. The consonant combination 'ndl' in Lindley sounds like 'nl', and thus pronouncing it as Linley. It works the same as Wiltshire to Wilshire, and Christmas to Chrismas.

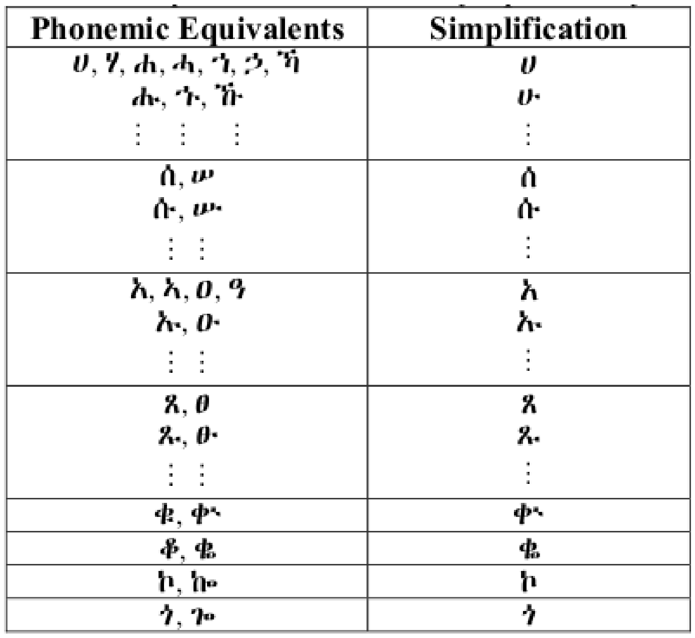
According to the results of Christian's study, there is a problem with how Soundex code starts with the first letter of the word, and the way it ignores vowels. This may produce misunderstandings, because *Askey* is A200, and *Haskey* is H200. Possible improvements for Soundex should include the addition of "*single letter*" rules where, initial letters should be ignored, all initial vowels should be coded to single codes, or finals consonants should be dropped.

2.2.2 Double Metaphone

A study by (Yacob, 2004) that uses Double Metaphone algorithm for the correction of Amharic orthography. Amharic orthography follows the rule "if a word sounds right when read aloud then it was rightly written". However, errors encountered on the Amharic spelling includes erros that are inherent from symbol redundancy, phonology-orthography disconnect, foreign language transcription, and typing system.

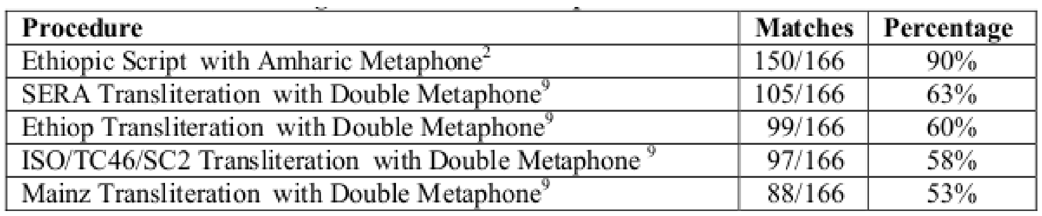
In the study, before implementing the metaphone algorithm for Amharic, Figure 2.1 conversion rules were created for the algorithm to be insensitive to the distinction between symbols but instead of sound.

On the double metaphone algorithm, two phonetic encodings will be generated were the frist encoded word is the most probable representative of



Phonemic Equivalents	Simplification
ሀ, ሃ, ሐ, ሓ, ኀ, ኃ, ኻ ሑ, ኁ, ኹ ⋮ ⋮ ⋮	ሀ ሁ ⋮
ሰ, ሠ ሱ, ሡ ⋮ ⋮	ሰ ሱ ⋮
አ, ኣ, ዐ, ዓ ኡ, ዑ ⋮ ⋮	አ ኡ ⋮
ጸ, ፀ ጹ, ፁ ⋮ ⋮	ጸ ጹ ⋮
ቁ, ቍ	ቍ
ቆ, ቈ	ቈ
ኮ, ኰ	ኮ
ጎ, ጐ	ጎ

Figure 2.1: Simplification of Phonetically Equivalent Syllables



Procedure	Matches	Percentage
Ethiopic Script with Amharic Metaphone[2]	150/166	90%
SERA Transliteration with Double Metaphone[9]	105/166	63%
Ethiop Transliteration with Double Metaphone[9]	99/166	60%
ISO/TC46/SC2 Transliteration with Double Metaphone [9]	97/166	58%
Mainz Transliteration with Double Metaphone[9]	88/166	53%

Figure 2.2: Words Matching Results for 166 Mispelled Forms of 116 Words

the target word while the second encoded word serves as an alternative under frequenty used English words. However, this study will limit more than two encodings because according to the researchers, many encodings might appear useful under Amharic writing conventions.

The study's Amharic metaphone algorithm was compared with the double metaphone algorithm with the same training sets. The gathering of the sample words was converted into an english representation so that the double metaphone can be applied. This method may affect the accuracy of the Double Metaphone.

The results at Figure 2.2 shows that the Amharic Metaphone achieved the highest accuracy rate amongst the another methods.

It is evident from the limited test conducted that higher matching rates are found for Amharic words when the Metaphone algorithm is adjusted to apply the rules of Amharic orthography in lieu of those of English. This positive result has held up as the test set of words has grown in number, a trend that is expected to continue. However, a much more thorough test with a larger corpus of materials should certainly be conducted.

Ultimately any approach to Amharic spelling correction is limited by the reliability of the reference used for canonical formations. The establishment of a comprehensive and authoritative lexicon for written Amharic would be the single most valuable resource towards the realization of this eventual goal.

2.3 Levenshtein Distance

A study of biological sequences using the Damerau-Levenshtein distance uses linear space algorithm to compute the distance between two strings and to find edit operation sequences. Their algorithm requires O(s minm,n + m + n) space whereas previous algorithm require O(mn) space. They have also developed a cache efficient and multi-core linear-space algorithm where the cache efficiency was analyzed using a simple cache model (Zhao & Sahni, 2017).

A study by (Hicham, 2012) present a new approach dedicated to correcting the spelling errors of arabic language. The method of the study is inspired from levenshtein algorithm and allows a finer and better scheduling than Levenshtein. The study modified the levenshtein distance between two words by taking into account the following matrices:

1. Matrix frequency of insertion error.
2. Matrix frequency of deletion error.

3. Matrix frequency of permutation error.

The result shows that compared to levenshtein, out of 190 erroneous words levenshtein has only 19 classified in the first position while the new approach correctly classified 119 words in the first position. the rest of 119 was distributed on the 2nd, 3rd, 4th, 5th .. 10th position. Statistically the study's method has proposed 62.63% of correct words in the first position against 10% for levenshtein.

Another Study by (Sarveswaran & Mahesan, 2015) integrated existing approach and new approach such as rule-base, crowdsourcing and suggestions generation using character level n-gram for spelling checker of Tamil language. In the proposed approach the levenshtein distance check each word in the dictionary whether it matches and flag the misspelled, the n-gram based technique is then used to generate possible suggestion of the misspelled word. 250,000 unique error-free words are included in their dictionary that have been collected in various sources using crowdsourcing, since it is very difficult to gather all words in Tamil Language. Required rules are written to get the appropriate suggestion by considering Canti check as well to identify the appropriate joining letter of two adjoined words. Furthermore, To reduce the search space, the dictionary has been divided into different files based on the first letter of the word. The result shows 85.77% accuracy when considering the suggestion generation. This result had been calculated by analysing the suggestions generated by the system for the words that are not in the dictionary.

2.4 Memoization

A study by (Johnson & Dörre, 1995) shows how constraints can be propagated using a memoizing parser in the same way as variable binding since some linguistic constraints cannot be all resolved during parsing at the location where they were naturally introduced. The study involves formalizing their approach to memoization and constraints within the general theory of Constraint Logic Programming (CLP) by Hohfeld and Smolka (1998). They also discussed how their method can be applied to a more standard chart parsing. Furthermore, their study extends the previous work of Dorre and Johnson by generalizing the methods used to arbitrary constraint system, which includes feature-structure constraints.

Another study by (Johnson, 1995), used memoization by defining a higher-order procedure memo which takes a function as an argument and returns a memoized version of it. This method stores the memo table as an association list resulting in a less than optimal implementation. Optimal performance would be probably by encoding string positions with integers in the parsing context allowing memo table look-up to be a single array reference.

A new technique by (Frost, 2003), enables a systematic and selective memoization of a wide range of algorithms. This technique can overcome disadvantages of previous approaches. This technique can help programmers avoid mistakes which can result in suboptimal use of memoization. The resulting memoized programs were amenable to analysis by using equational reasoning. According to Frost's study, the programmers will be provided with a function memoize which can be used to *memoize* selected parts of a program. When the *memoized* parts of the program is executed then the function memoize will take care of updating and using the memo tables.

Most of the study that uses large lexicon produces higher percentage of accuracy and low percentage of error rate this shows that most of the spelling checker depends on the lexicon used however SoundEx produces many suggestion words but most of this suggestion are inefficient because of many possible suggestions using the code. Also, most of the spelling checkers uses N-grams in producing suggestions because of its accuracy but the lengths of the words in the dictionary must always be considered when choosing what N-gram will be used - bi-gram, tri-gram, four-gram, or five-gram.

Chapter 3

Theoretical Framework

The discussion below includes topics on Natural Language Processing, Metaphone, Levenshtein Distance, and Memoization. These discussions help readers to understand the terms that are used in this study.

3.1 Natural Language Processing

According to (Chowdhury, 2003), Natural Language Processing or NLP is the area of research that explores how computers are used to understand and manipulate natural language texts so that it can do useful things.

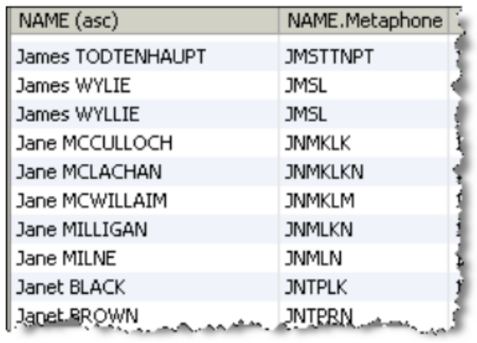
Another definition states that NLP is the computerized approach to analyzing texts that are based on both sets of theories and sets of technologies (Liddy, 2001).

Researchers in NLP aim to gather information on how human beings understand and use language so that there will be appropriate tools and techniques will be developed to make computer systems understand and manipulate natural language to perform desired tasks. Machine translation, natural language text processing and summarization, user interfaces, multilingual and cross language information retrieval (CLIR), speech recognition, artificial intelligence and expert systems, and many more are only some of the applications of NLP (Chowdhury, 2003).

3.2 Metaphone

Metaphone is an english phonetic algorithm published by Lawrence Philips in 1990. Metaphone processor represent the string value into a code based on its pronunciation. The processor allows to specify the maximum length of the Metaphone code (up to a maximum of 12 characters). An example shown at Figure 3.1 transform the NAME attribute with the use of default maximum length of 12 characters.

Lawrence Philips later on produce a new version of metaphone called Double Metaphone. It is called "Double Metaphone" because it returns a both a primary and a secondary code for a string.



NAME (asc)	NAME.Metaphone
James TODTENHAUPT	JMSTTNPT
James WYLIE	JMSL
James WYLLIE	JMSL
Jane MCCULLOCH	JNMKLK
Jane MCLACHAN	JNMKLKN
Jane MCWILLAIM	JNMKLM
Jane MILLIGAN	JNMLKN
Jane MILNE	JNMLN
Janet BLACK	JNTPLK
Janet BROWN	JNTPRN

Figure 3.1: Metaphone

3.3 The Filipino Orthography

In pronouncing borrowed words or "Hiram na Salita" from its original spelling the following rules should be followed:

3.3.1 Maintaining the structure of the original word

The letters, both vowels and consonants, can represent more than one sound, for example:

1. **'c' - 'k', 's', or 'č'.** "casa" = ('ka.sa), "cello" = ('če.low), and "ice" = ('?ays).
2. **'j' - 'h'.** "jack" ('jak), and "jai alai" ('hay ?a.'láy)
3. **'x' - 's', or 'ks'.** "extra" ('?eks.tra), and "xylophone" = ('say.lo.fown).
4. **'f' - 'p'.** "father" - ('pa:.der)

5. **’v’ - ’b’.** “visa” (vi.sa) - (bi.sa)

6. **’z’ - ’s’.** “zoo” (zu) - (su)

7. **’æ’ - ’a’.** “map” (mæp) - (map)

8. **’ow’ - ’o’.** “goal” (gowl) - (gol)

9. **’i:’ - ’i’.** “brief” (brif) - (brip)

10. **’u:’ - ’u’.** “shoot” (šut) - (šut)

3.4 Levenshtein Distance

Levenshtein Distance between two strings is the minimal number of insertions, deletions, and substitutions of one character for another that will transform the source string into the target string. The distance is the number of deletions, insertions, or substitutions required to transform the source string into the target string. The Minimum Edit Distance uses sequence of edits represented by Figure 3.2.

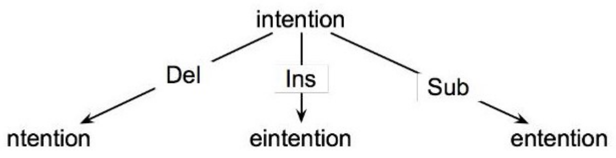


Figure 3.2: Minimum Edit Distance

The greater the Levenshtein distance, the more different the strings are. Furthermore, the Levenshtein Distance Algorithm has been used in Speech Recognition, Plagiarism Detection and Spell Checking.

3.5 Memoization

The term "memoization" was introduced by Donald Michie in the year 1968. It's based on the Latin word memorandum, meaning "to be remembered". It's not a misspelling of the word memorization, though in a way it has something in common.

Memoization is a technique used in computing to speed up programs. This is accomplished by memorizing the calculation results of processed input such as the results of function calls. If the same input or a function call with the same parameters is used, the previously stored results can be used again and unnecessary calculation are avoided. In many cases a simple array is used for storing the results, but lots of other structures can be used as well, such as associative arrays, called hashes in Perl or dictionaries in Python (Stonelen, 2006).

In this study, when the same word occurs again in the document the system will no longer need to check if this input data is valid word or invalid word instead the system will called the data store by memoization.

Chapter 4

Methodology of the Study

In this chapter, the following are discussed: Conceptual Framework, Data gathering, Storing Data to Database, Spell Checker, Spell Corrector, Post-processing, Evaluation and the Tools and Technology.

4.1 Conceptual Framework

The will system have five stages - preprocessing, dictionary lookup, extracting suggestions, postprocessing and evaluation. First, the *preprocessing of the dictionary* involves the grouping of all words according to their similar encodings and the *preprocessing of the misspelled word* involves the checking if the input appeared recursively or not, and the conversion of the misspelled word to double-metaphone format referred as *meta-word*. Then, the *dictionary lookup* involves matching the misspelled word to the *Dictionary*. The

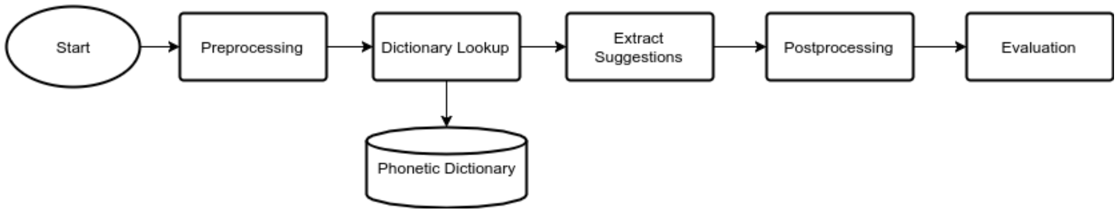


Figure 4.1: System Architecture

Dictionary contains double-metaphone formats referred as *code* and their corresponding words with the format
{'code': ['word1', 'word2', ..., 'wordn']}. In the *Extract Suggestions Module* module, the matched *code* from the *meta-word* will be retrieved with its corresponding words - the suggestions. After that, the suggestions will be ranked. A maximum of 10 suggestion will be set and these suggestions can only be displayed. Then in the *postprocessing* stage, there will be a client-side notification so that the user will be able to choose if they want to replace the source word or not. Finally, the system will be evaluated according to its precision. Figure 4.1 illustrates the stages of processing the words in our system.

4.2 Data Gathering

The researchers will gather documents that merely consist of texts. The document should only have a maximum number of 250 words. Some of these documents will be gathered from the Filipino Department, College of Arts

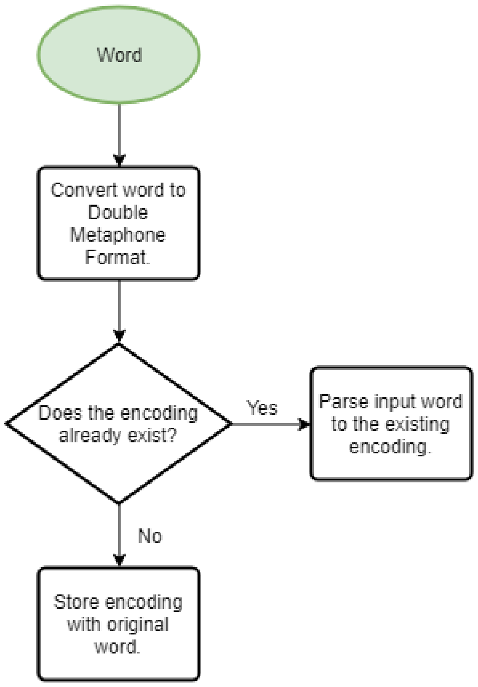

Word
Convert word to Double Metaphone Format.
Does the encoding already exist?
Yes
Parse input word to the existing encoding.
No
Store encoding with original word.

Figure 4.2: Process Flowchart of Dictionary Preprocessing

and Social Sciences. The study will also utilize the data from a Filipino dictionary for Aspell.

4.3 Preprocessing

4.3.1 Misspelled Word

Figure 4.2 is the visual representation of the process which will be discussed further in this section.

The data of the dictionary is specifically formatted in the manner where *code* and their corresponding words called *wordn* will be coded into {'code': ['word1', 'word2', ..., 'wordn']}. First, the word will be encoding to its double metaphone format, referred as *code*, where Table 4.1 shows the Metaphone values of each characters based on the Filipino Orthography. Then system will first check if the encoding already exists or not. If the encoding doesn't exists, then the encoding will be stored with the input word in the format {'code': ['word']}. But if the encoding already exists, then the word will be parsed to the encoding in the format {'code': ['word', 'new word']}.

The vowels are only used in the encoding when it is at the beginning of the word and compared to Metaphone, Double Metaphone returns both the primary and secondary code for a string. For example, the letter *K* has two encodings - *K* and *Q*. So, given a word **kolehiyo**, the system will generate two encodings, *KLHY* as primary and *QLHY* as secondary. Also, given a word beginning with a vowel, *alapaap*, the system will generate the encodings *ALPP* and *ALFF*.

4.3.2 Misspelled Word

Figure 4.3 is the visual representation of the process which will be discussed further in this section.

The *commonwords* database contains the memoized data from the previous input of the system. The memoized data will be removed from the *Dictionary* and it will be transfered to the *commonwords* database since the misspelled word input is first matched in the *commonwords* database and if

Value	Letters
a	'a', 'e', 'i', 'o', 'u'
b	'b', 'v'
c	's', 'k'
d	'd'
f	'p', 'f'
g	'g'
h	'h'
j	'j', 'h', 'dy'
k	'k','q'
l	'l'
ll	'ly', 'y'
m	'm'
n	'n'
Ñ	'ny'
ng	'ng'
p	'p', 'f'
q	'kw', 'ky', 'k'
r	'r'
s	's'
t	't'
v	'b', 'v'
w	'w', 'ua', 'ue', 'ui', 'oa', 'au', 'ou'
x	'ks', 's'
y	'y', 'ia', 'ie', 'io', 'iu', 'ea'
z	'z', 's'

Table 4.1: Phonetic Values Table

Algorithm 1 Double Metaphone Algorithm

DoubeleMetaphone(s) transforms string s of length *k(s)* into the Double Metaphone primary and secondary keys, t_1 and t_2 respectively.
The major steps of the algorithm are as follows:

1. Set the phonetic values.
2. Initialize input string.
3. Transform input string into its Double Metaphone encoding.

We now examine these steps in detail.

1. Set the phonetic values

   Table 4.1 contains the phonetic rules, denoted as rn, for all 26 letters of the Filipino alphabet, denoted as fm. Where m is the number of the Filipino alphabet, and n is the number of rules in each alphabet.Take note that both rn and fm should in uppercase.

   (a) Set $f_m \leftarrow (r_1, r_2, ..., r_n)$

   the number of rules for every alphabet is len(f_m) >= 1.

2. Initialize input string.
   This step normalizes the input string, denoted by s, into its uppercasing.

   (a) input = s
   (b) input.upper()

3. Transform input string into its Double Metaphone encoding.
   To produce the primary and secondary keys, denoted by t1 and t2 respectively, the input string s will first go over a **while** loop, then a succession of nested **if-else** loops. Below is the phonetic rule for the letter "B":

   (a)

```
if symbol is "B" then
    $t_1 \leftarrow$ "B"
    $t_2 \leftarrow$ "V"
    Increment current

else
    $t_1 \leftarrow$ "B"
    $t_2 \leftarrow$ "V"
    Increment current
    continue

end if
```

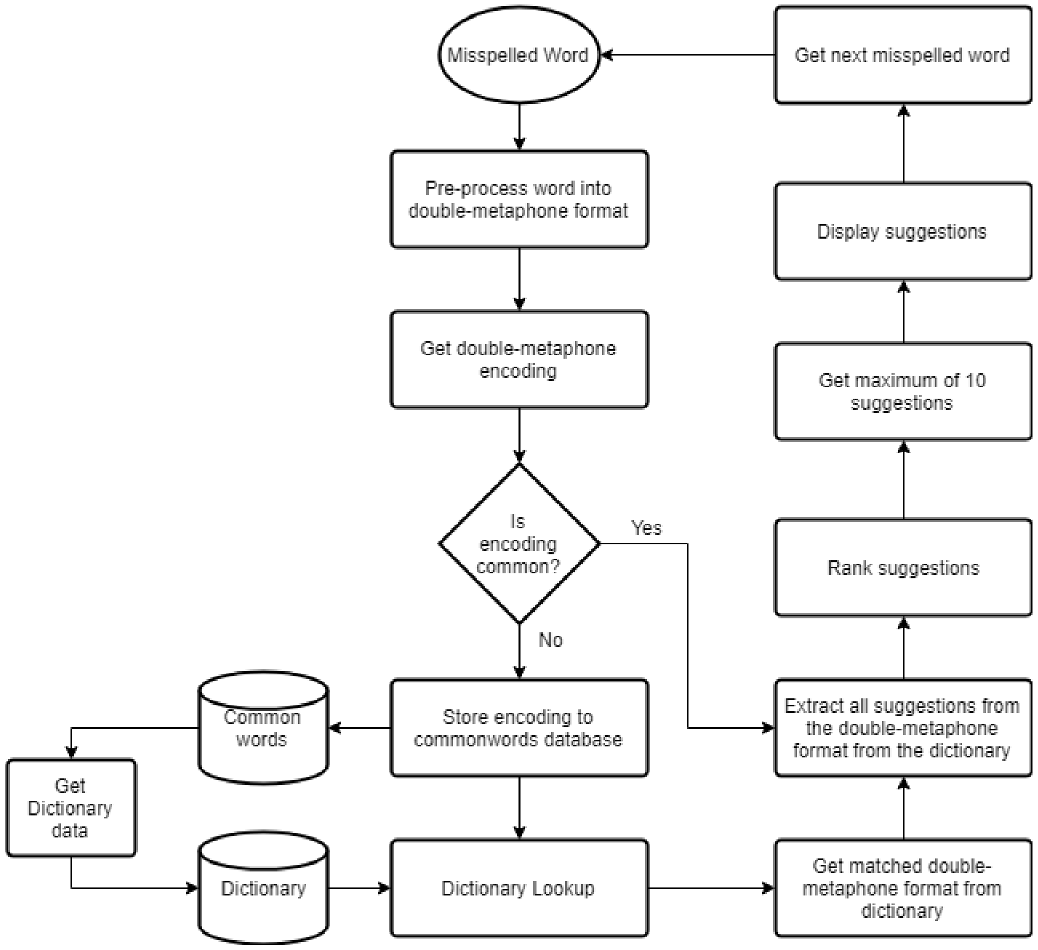
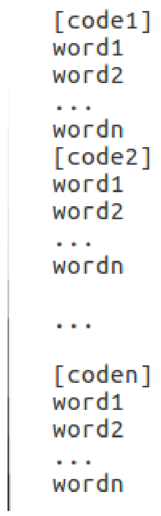

Misspelled Word
Pre-process word into double-metaphone format
Get double-metaphone encoding
Is encoding common?
Yes
No
Store encoding to commonwords database
Common words
Get Dictionary data
Dictionary
Dictionary Lookup
Get matched double-metaphone format from dictionary
Extract all suggestions from the double-metaphone format from the dictionary
Rank suggestions
Get maximum of 10 suggestions
Display suggestions
Get next misspelled word

Figure 4.3: Process Flowchart of Misspelled Word Preprocessing



```
[code1]
word1
word2
...
wordn
[code2]
word1
word2
...
wordn

...

[coden]
word1
word2
...
wordn
```

Figure 4.4: Dictionary Layout

the misspelled word is not common, then it will be matched to the *Dictionary*.

If the memoized data wont be transfered to the *commonwords* database, then the input will be compared again to (in worst case) more than 50% of the *Dictionary* data that is already in the *commonwords* database. Thus, this method is necessary.

The *dictionary* and *commonwords* database's data will inputted into the format shown at Figure 4.4

The misspelled word input will be converted to its double-metaphone format.

4.4 Dictionary Lookup

The misspelled word, which is in double-metaphone format called *meta-word*, will be matched in the dictionary. If a match is found, then the code with its corresponding word will be retrieved.

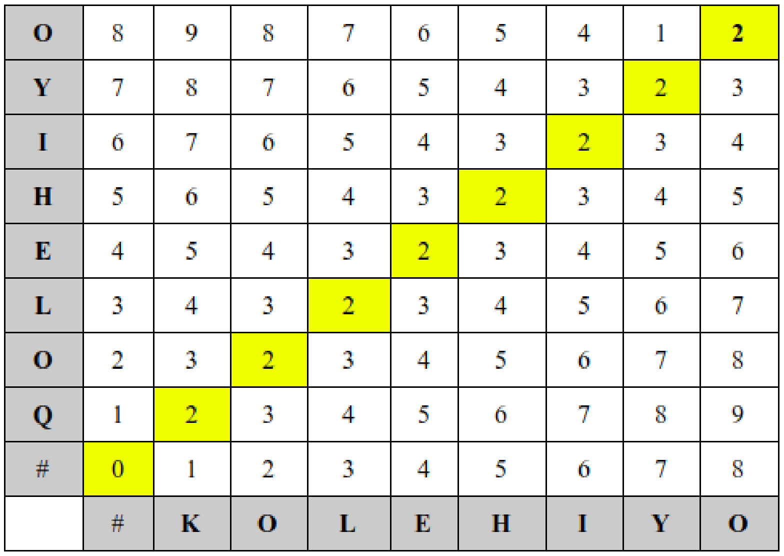
Given a misspelling *qolehiyo*, the encoding that will be generated will be *KLHY* and *QLHY*. The system will retrieve matches for these encoding with its corresponding words from the dictionary.

4.5 Extract Suggestions

The corresponding words of the code on the dictionary will be extracted. These corresponding words will be the suggestions. The suggestions will be ranked by using Levenshtein Distance. After ranking, only a maximum of 10 suggestions will be retrieved.

In finding the edit distance between the source word *qolehiyo* and a target word *kolehiyo*, we will be using the algorithm below where the results are visualized Figure 4.5.

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+1, & \begin{cases} 2, & \text{if} S_1(i) \neq S_2(i) \\ 0, & \text{if} S_1(i) = S_2(i) \end{cases} \end{cases}$$



O	8	9	8	7	6	5	4	1	**2**
Y	7	8	7	6	5	4	3	2	3
I	6	7	6	5	4	3	2	3	4
H	5	6	5	4	3	2	3	4	5
E	4	5	4	3	2	3	4	5	6
L	3	4	3	2	3	4	5	6	7
O	2	3	2	3	4	5	6	7	8
Q	1	2	3	4	5	6	7	8	9
#	0	1	2	3	4	5	6	7	8
	#	**K**	**O**	**L**	**E**	**H**	**I**	**Y**	**O**

Figure 4.5: The Edit Distance Table

4.6 Postprocessing

An instance were the *source word* is a noun and flagged as a misspelled word since it is not located in the dictionary. To solve this problem, we propose a *client-side notification* were the user will have to choose whether to treat the *source word* as a *misspelled word* and replace it with the *correct word* or not. If the *source word* is indeed a misspelled word but it is not in the dictionary, then the *source word* will be added to the *commonwords* dictionary along with its corresponding word suggestions.

Algorithm 2 Levenshtein Distance

LevenshteinDistance(s,r) determines the edit distance between two string, *s* and *r*, whose lengths are *k(s)* and *k(r)*. The edit distance can be acquired by setting a value for every insertions, deletions, and substitutions.
The major steps of the algorithm are as follows:

1. Initialize the matrix.
2. Compute the Edit Distance.

We now examine these steps in detail.

1. Initialize the matrix.
   The matrix will have a *k(s) x k(r)* size. Also, the row and column of the matrix will indexed from 0 to *k(s)* and 0 to *k(r)* respectively.

   (a) Set $matrix \leftarrow [k(s)+1][k(r)+1]$

2. Compute the Edit Distance.
   Each letters (row-wise, and column-wise) will be compared. If two letters are the same, then the value at index *[x,y]* will be the minimum of the values at index *[x-1, y]+1*, index *[x-1, y-1]*, and index *[x, y-1]+1*. Otherwise, the values at index *[x,y]* is the minimum of the value at index *[x-1, y]+1*, index *[x-1, y-1]+1*, and index *[x, y-1]+1*. After traversing to all letters, the edit distance will be the value at index *[k(s)+1, k(r)+1]*.

   (a)
```
for $x \leftarrow 1...k(s)+1$ do
    for $x \leftarrow 1...k(s)+1$ do
        if index $[x-1,0] \leftarrow$ index $[0,y-1]$ then
            matrix $[x,y] \leftarrow$ minimum of (
                $[x-1,y]+1$,
                $[x-1,y-1]$,
                $[x,y-1]+1$)
        else
            matrix $[x,y] \leftarrow$ minimum of (
                $[x-1,y]+1$,
                $[x-1,y-1]$,
                $[x,y-1]+1$)
        end if
    end for
end for
```

4.7 Evaluation

In this study the evaluation of spelling checker is based on ISO 9126 standards. For our proposed metrics, we used the Suggestion Measures from the combined the metrics of TEMAA, Starlander & Popescu-Belis , and Van Zaanen & Van Huyssteen where we describe the following:

1. True positives (Tp): valid words recognised by the spelling checker.

2. True negatives (Tn): invalid words recognised by the spelling checker.

3. False negatives (Fn): valid words not recognised by the spelling checker.

4. False positives (Fp): invalid words not recognised by the spelling checker.

4.7.1 Suggestion Measures

According to (Van Huyssteen, Eiselen, & Puttkammer, 2004), suggestion adequacy or SA is the spelling checker's ability to produce relevant suggestions for all incorrect words that are flagged by the spelling checker or the true negatives. The SA of a spelling checker should only be based on true negatives, and not on all negatives, since the aim is to determine how well the spelling checker can suggest corrections for incorrect words.

The following scoring system is derived from Paggio & Underwood, and Van Zaanen & Van Huyssteen's scoring system:

1. Correct suggestion is the first suggestion = 1 (CS 1).

2. Correct suggestion is a visible suggestion = 0.5 (CS 2).

3. All corrections are incorrect = -0.5 (IS).

4. No suggestions = 0 (NS).

Therefore, for each correct suggestion, the spelling checker scores 1, and for each visible correct suggestion 0.5. If the spelling checker offers only incorrect suggestions, it is penalised with -0.5. 4 However, if the spelling checker does not offer any suggestions, it scores 0.

$$SA = \frac{\sum_{k=1}^{n} S_k}{N_{Tn}} \tag{4.1}$$

Where S is the score for every suggestion, and N is the total number of True Negatives (Tn).

4.7.2 Overall Performance Measure

The run-time of the spelling checker that uses Double Metaphone and Levenshtein distance in producing spelling suggestions will be compared to the run-time of the spelling checker which is only using Levenshtein distance. Python's built-in function, *time*, will be used to obtain this method.

```
import time

    start = time.time()
    spellchecker code
    finish = time.time() - start
```

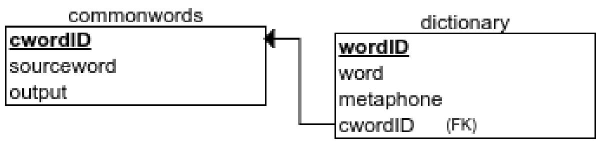


Figure 4.6: Relational Diagram

4.8 Tools and Technology

4.8.1 Python

The researchers will be using Python programming language because it provides software for multiple platforms and libraries for levenshtein distance.

4.8.2 MySQL

The researchers will be using MySQL as the database of the system because it uses a fast thread-based memory allocation system. Figure 4.6 shows the relationship of the database to the system.

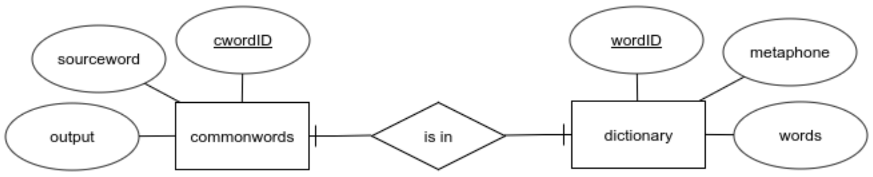


Figure 4.7: ERD

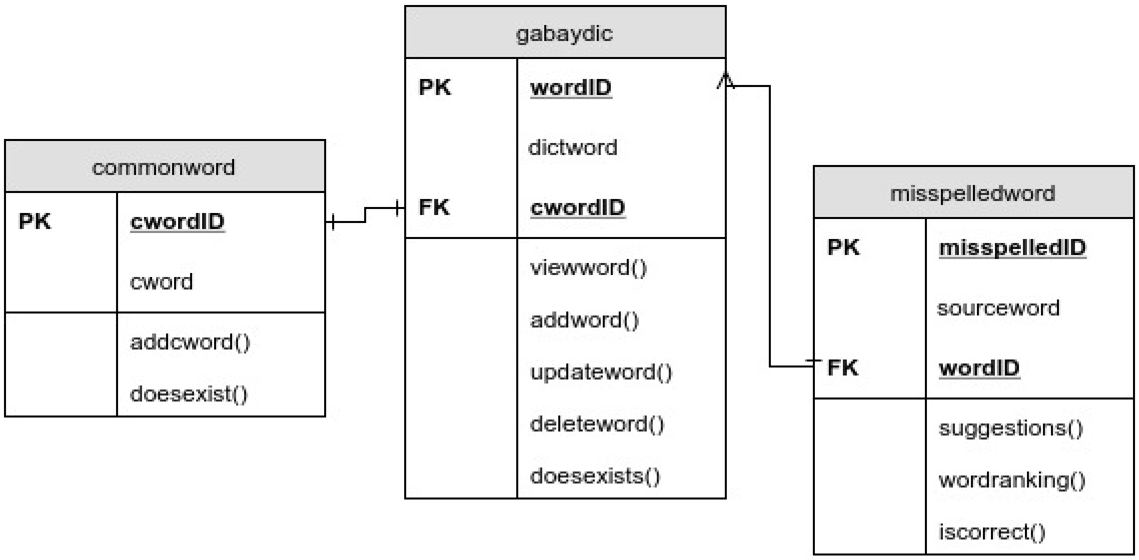


Figure 4.8: Class Diagram

Chapter 5

Schedule

Activities	**Due of Date**
Final Revision of Proposal	May 27-31
Implementation	June to September
Gather Training Sets, Populate Database	October 1-12
Chapter 5: Evaluation; and Revision	October 13-20
Chapter 6: Discussion and Conclusion; and Revision	October 21-27
Overall Revision	November 12-16
Final Revision	November 17-2
Submission of Final Paper	November 26-30

Table 5.1: Research Schedule

References

Atkinson, K. (n.d.). *How aspell works.* `https://www.aspell.net`.

Atkinson, K. (2016). *Gnu aspell.* `https://www.aspell.net`.

Blank, D. (2012). Spelling checking algorithms.

Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. *Communications of the ACM*, *20*(10), 762–772.

Cheng, C., Alberto, C. P., Chan, I. A., & Querol, V. J. (2007). Spellchef: Spelling checker and corrector for filipino. *JOURNAL OF RESEARCH IN SCIENCE, COMPUTING, AND ENGINEERING*, *4*(3).

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, *37*(1), 51–89.

Christian, P. (1998). Soundex-can it be improved? *Computers in Genealogy*, *6*, 215–221.

Frost, R. (2003). Monadic memoization towards correctness-preserving reduction of search. In *Conference of the canadian society for computational studies of intelligence* (pp. 66–80).

Gredler, G. R. (2002). Preventing reading difficulties in young children. *Psychology in the Schools*, *39*(3), 343–344.

Gucasan, N. (2017). *Filipino. tagalog. pilipino.* `https://www.tagaloglang.com/filipino-tagalog-pilipino/`.

Hicham, G. (2012). Introduction of the weight edition errors in the levenshtein distance. *arXiv preprint arXiv:1208.4503*.

Ispell. (2006). `https://www.gnu.org/`.

Johnson, M. (1995). Memoization in top-down parsing. *Computational Linguistics*, *21*(3), 405–417.

Johnson, M., & Dörre, J. (1995). Memoization of coroutined constraints. In *Proceedings of the 33rd annual meeting on association for computational linguistics* (pp. 100–107).

Knuth, D. E., Morris, J. H., Jr, & Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM journal on computing*, *6*(2), 323–350.

Kukich, K. (1992). Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*, *24*(4), 377–439.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, pp. 707–710).

Liddy, E. D. (2001). Natural language processing.

Moats, L. C. (2005). How spelling supports reading. *American Educator*, *6*(12–22), 42–43.

Ndaba, B. (n.d.). Afrispel: An isizulu spellchecker.

Philips, L. (2000). The double metaphone search algorithm. *C/C++ users journal*, *18*(6), 38–43.

Sarveswaran, K., & Mahesan, S. (2015). Hierarchical tag-set for rule-based processing of tamil language. *International Journal of multidisciplinary Studies*, *1*(2).

Sellers, P. H. (1980). The theory and computation of evolutionary distances:

pattern recognition. *Journal of algorithms*, *1*(4), 359–373.

Stanier, A. (1990). How accurate is soundex matching. *Computers in Genealogy*, *3*(7), 286–288.

Van Huyssteen, G. B., Eiselen, E., & Puttkammer, M. (2004). Re-evaluating evaluation metrics for spelling checker evaluations. In *Proceedings of first workshop on international proofing tools and language technologies* (pp. 91–99).

Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, *21*(1), 168–173.

Yacob, D. (2004). Application of the double metaphone algorithm to amharic orthography. *arXiv preprint cs/0408052*.

Zhao, C., & Sahni, S. (2017). Efficient computation of the damerau-levenshtein distance between biological sequences. In *Computational advances in bio and medical sciences (iccabs), 2017 ieee 7th international conference on* (pp. 1–1).

Zobel, J., & Dart, P. (1995). Finding approximate matches in large lexicons. *Software: Practice and Experience*, *25*(3), 331–345.

Zobel, J., & Dart, P. (1996). Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th annual international acm sigir conference on research and development in information retrieval* (pp. 166–172).