May 2016

Writing English Sentences More Effectively By Avoiding Arabian Students' Typical Mistakes

Alaa Mohammed Alsharif
University of Wisconsin-Milwaukee

WRITING ENGLISH SENTENCES MORE EFFECTIVELY BY AVOIDING ARABIAN STUDENTS' TYPICAL MISTAKES

by

Alaa Alsharif

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

May 2016

ABSTRACT

WRITING ENGLISH SENTENCES MORE EFFECTIVELY BY AVOIDING ARABIAN STUDENTS' TYPICAL MISTAKES

by

Alaa Alsharif

The University of Wisconsin-Milwaukee, 2016
Under the Supervision of Professor Susan McRoy

In Arabic speaking countries like Saudi Arabia, English is considered as the most important second language to be taught and used. Unfortunately, a sizeable percentage of students there appear to still have significant difficulty learning English, possibly due to the difficulty in finding sufficiently qualified teachers. This type of problem is self-perpetuating since the taught students might become weak teachers in the future also. This thesis aims to address the problem of helping Arabic students to improve their writing in English and to help them learn so that they will make fewer mistakes in the future and possibly become better teachers themselves. It focuses on creating methods to find the most typical mistakes made by those Arabian students in their writing, mistakes which were determined by the author from both self-observation and a review of related research findings (The author also saw these mistakes in the sentences used by subjects who tried pilot versions of the software). The result of this work is usable software that is able to detect, correct, and provide grammatical rules related to the most common mistakes found in the written sentences of the target Arabian students, when the sentences are in the present tense. These types are errors related to the following rules: (1) letters capitalization rules, (2) adj-noun ordering in the sentence, (3) proper use of the verb *to be*, (4) punctuation

placement rules, (5) the use of the articles “a” and “an” within a sentence, and (6) rules for the possessive case.

The software was evaluated using the author’s observation on the use of the software by 22 Arabian students and by letting them afterwards to complete a usability and usefulness survey. The results of the evaluation suggest that Arabs will mostly like how the software treats punctuation placing errors. Students also advised the author that it would be beneficial for the software to address a broader range of typical mistakes.

This work is the first to create software specifically for Arabic students of English to help them to find their grammatical errors, provide suggested correction, and teach the student the grammatical rules needed to correct his/her sentence.

Dedicated to my mother in the first place,

father,

husband,

brothers,

friends,

and all of my family members who are still alive or died recently

for encouraging me even when I was thinking to quit completing my studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ESL - English as a Second Language.

GUI - Graphical User Interface.

NLTK - the Natural Language Toolkit.

UAE - United Arab Emirates.

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this thesis, a great many people have contributed to its production. I owe my gratitude to all those people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

My deepest gratitude is to my advisor, Prof. Susan McRoy. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. Mrs. McRoy taught me how to question thoughts and express ideas. Her patience and support helped me overcome many crisis situations and finish this thesis. I hope that one day I would become as good an advisor to my students as Mrs. McRoy has been to me.

Many friends have helped me stay sane through these difficult years. Their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me. I am also grateful to the Arabian students that helped me evaluate my work.

I'm also thankful for all of the UW-Milwaukee staff in Computer Science department who helped me through their experience.

Finally and most importantly, none of this would have been possible without the love and patience of my family. My immediate family to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express

my heart-felt gratitude to my family. My extended family has aided and encouraged me throughout this endeavor.

Chapter 1

Introduction

1. Introduction

1.1 Main Ideas

When students travel to obtain an education they face many challenges, including needing to write scholarly essays or reports in a language that may be very different from their native language. These differences can cause them to make errors in grammar or style, which could place their academic success at risk. This thesis aims to address this problem by creating new software that will help prevent or correct the typical mistakes that Arab students make when writing formal documents in English.

The potential impact of this thesis is high because of the huge number of Arab students in the United States, dispersed over many states and cities. An important goal is that the software be perceived as beneficial to the students who are its target audience. The author feels she has a good understanding of what would be seen as beneficial, because she is herself an Arabic student in an English country.

Arab students are supposed to read, write, listen, and speak English fluently to pass their courses successfully. Each one of those four skills is very important and also complex. For example, reading is an English skill that requires the student to scan the paragraphs, skim the book topics, and comprehend the details and information presented in those paragraphs. Unfortunately, time limitations forced the work in this thesis to address only one of them. The author chose to solve the most noticeable problems related to English writing due to repeated requests from Arab students to have something helpful for them related to improving their

English writing.

1.2 Motivations

Growing up in Saudi Arabia where the mother language is Arabic and the second language is English helped the author to understand the kind of work that needed to be done to help Saudi students with their English writing. Until this point of time in Saudi Arabia, students are still not properly educated in their schools about how to use the English language in their lives and studies. This problem of being not educated well in the English language has increased dramatically in the last five years and is especially noticeable in all of the big cities. There are many reasons for this growth; the next paragraphs will explain the most significant ones which have motivated the work on this problem.

After finishing high school, any Saudi student who wants to complete his/her studies in the field of engineering, medicine, or science as an undergraduate student is required to pass about eight months of studying the English language. The Ministry of Education in Saudi Arabia requires students to pass these English training courses because the universities focusing on these fields of studies are using English in their curriculum. However, imagine how difficult it must be for a beginner student to become a fluent English speaker, writer, listener, and reader in only one year with only short English lessons daily. Most of the students are able to pass the training course because they already know in advance what they will be asked for in their final exams and what they should answer. However, many of them will still not really be fluent in the English language or even good users of English. Some students will struggle and suffer greatly until they get a chance to learn English more properly by living in an English country for a while

where they can learn the language truly.

The second significant reason for the weak English skills among many Arab students is rooted in the huge number of scholarships which are paid by the government to some students who have special characteristics. Students use the scholarships to travel to English-speaking countries to finish their bachelor's degree, masters, Ph.D., or all of them together. The life and the study for those students with scholarships are even harder than those who attend English-language institutions in their home country, because there are often higher expectations for their English skills. However, generally these students are also inadequately prepared. Writing, being perhaps the most essential skill for them, was thus chosen as the primary topic for the proposed software.

1.3 Contributions Toward Solving the Problem

From the motivations provided earlier in this chapter, the author became inspired to create an English writing program or an application that checks the grammar of the written sentences. The application is intended to help Arab students to learn the correct structure of writing sentences in English by avoiding the most typical mistakes done by Arab students. The approach makes use of, the differences between the structure of the English sentences and the structure of Arabic ones, since they are almost completely different and the opposite. The program aims to teach Arabic students the basic correct sentence structure in English by giving them suggestions on how the written sentence should look and explaining the grammatical rules behind the suggested structure.

Creating an application that gives Arab students only instructions about the right structure of the English sentences might seem to be boring, unattractive, and faraway of being a programmed application. The author addresses this risk by giving students the ability to write their own sentences and then check them for errors. To understand what are the typical mistakes, the research will involve a survey of past work. The typical mistakes documented previously will be the basis of the program and the main concern that the program will try to correct.

Another approach taken in this work was to make observations of the common mistakes of Arabian friends in their English writings and conversations in their basic daily life. Observing them was a good method to prove and validate the typical mistakes of Arabic students in writing English which are found by the first approach.

The results of the research and observation in pointing out the weakness in Arabs' English writing skills directed the work to the key mistakes that should be corrected in Arabs' sentences using the software.

1.3.1 Summary of Contribution Findings

From observation and research, the most common mistakes of Arab students in writing in English were found to be more diverse than what could reasonably be covered and detected by the new software. For this reason, the software will solve only the most common mistakes.

Specifically, the software will be able to detect mistakes like: the use of articles such as "a" and "an", the genitive ordering of nouns, use of punctuation, proper

capitalization of letters including the first letter of the first word in the sentence and the proper nouns, as well as the use of verb to be in the sentence. Note that the program focuses on the present tense more than the others.

Although the program may miss some corrections, users will never get wrong suggestions because mistakes that were not detectable will be ignored. It will never say that the sentence is correct, because it was programmed only to detect the specific mistakes provided in the previous paragraph and nothing else. Therefore, although the sentences will be improved, we cannot yet say that the sentence is error free.

In the next chapter, we will consider related work done in this field and past insights into the right structure of written English sentences. After that, the remaining chapters will answer the following questions: how effective can a software system be in identifying and correcting typical errors, how usable, useful and satisfying will Arabic students find such a system, and what design measures are used to create the interface.

Chapter 2

Background

2. Background

2.1 Consensus Opinions

Many experts have agreed that English language and Arabic are very different from each other. This makes it challenging for the Arab speakers to learn English fluently. Since the rules in forming sentences in Arabic is different in English, most Arab students still need to unlearn the rules in writing sentences and relearn it in a new when writing in English. There is a consensus about this hardship among Arab speakers who are learning English and the need for support or a program that will help these students identify their mistakes and correct them.

Arabic language is from Semitic language family while English is from Germanic language. Linguists know that the grammar of these two languages are very different and there are parts of speech in English language that is not part of Arabic speech ("The Differences between English and Arabic"). Therefore, it is very understandable for the beginners to be confused and they need help to correct their common mistakes.

In a study conducted by Haifa Al-Buainain, an associate professor at the Department of Foreign Languages at Qatar University, 40 exam scripts were analyzed from the students taking their first Writing Course in English at the university. Using error analysis as a method to analyze the students' writing, Al-Buainain found out that the errors of students "systematic and classifiable" (1). To be more specific, it was found out in this study that Arab students who are beginners in the English writing have concerns about "structure, selection of vocabulary items, spelling, punctuation, organization, adequacy of ideas and variety of sentence structure, logic and strength of argument" (Al-Buainain 3). In others, the problem is not just with grammar but

also with selection of the appropriate words to be used and the strength in writing sentences to create claims or arguments.

When it comes to grammar, the most common problem of the ESL (English as a Second Language) Arab students is the proper use of verbs using the correct tense. For instance, one student in the study conducted by Al-Buainain wrote “They always shouting and open my room” (6). Clearly, this sentence has errors in the use of present continuous form of the verb as well as the present tense of the verb. Due to these errors, the sentence also had a problem with parallelism. Once the verbs are parallel, the sentence will be better as a sentence written as “They always shout and open my room.” Although the sentence may still lack some clarity, writing it that way makes it more grammatically correct.

Another common mistake of the ESL Arab students in writing is the omission of the verb *to be*. For instance, one student wrote “I interested” instead of writing “I am interested” (Al-Buainain 6). Since the Arabic language has no *to be* or its equivalent in its language, this omission mistake is common among Arab ESL students.

Furthermore, the lack of subject/verb agreement is also rampant among Arab students when writing in English. Since Arabic language has no subject/verb agreement, most Arab students commit mistakes in making singular subject agree to singular verb and plural subject agree to plural verb. For instance, one student in the same study wrote “Their market and shopping centres has…” (Al-Buainain 7). Since the subject in the given sentence is plural, the verb should be *have* instead of *has*. Other common errors explained in the study conducted by Al-

Buainain include the use of articles, writing fragments, noun modifiers, countable and uncountable nouns, and preposition.

Another study conducted by Taiseer Mohammed Y. Hourani - this time from United Arab Emirates (UAE) – also aimed to know the common mistakes made by Arab students. Since Arabic is also the language in UAE, this should be the same with the common concerns faced by Arab students in Saudi Arabia when writing English sentences. Hourani found similar grammatical errors which are similar to what Al-Buainain found in his study. These errors include "passivization, verb tense and form, subject-verb agreement, word order, prepositions, articles, plurality, and auxiliaries" (Hourani).

With the similar findings of two different studies done, including the observation I had with the Arab students around me, it can be concluded that the findings of these researchers are validated with the observations I had and the consensus among the study results conducted among different groups of ESL Arab students who are writing in English. The articles mentioned include the use of "*a"* and "*an"* while the word order include the order of adjectives and nouns in the sentence which is sometimes confusing to the students.

Meanwhile, Lina Gomaa, a language instructor who taught at Beloit College in Wisconsin and at Misr International University in Cairo also shared her opinion about the common mistakes that Arab students make when writing English sentences. As an educator, she based her statement from her own experiences. According to her, there are five major trouble spots for Arabic ESL students. These five major trouble spots include but not limited to (1) run-on sentences, (2) redundancy, (3) Arabish, (4) punctuation, and (5) writing organization. When it

comes to run-on sentences, what Gomaa means is that students tend to write endless sentences without using period. This is a typical mistake among Arab students since most of them do not know when to put full stop. Moreover, the Arabish words is still present in papers of Arab students since they directly translate Arabic words to English at times just like in a sentence like "infection spreads by peace with hand" (Gomaa). In Arabic language, this is easily understandable; however, when directly translated to English word-for-word, it does not make a lot of sense anymore. In addition, the use of punctuation is also a common mistake among Arabic students since Arabic language has "less limitations in the use of commas and periods than English" (Gomaa). This coincides with the study conducted by other researchers saying that most Arab students have problems in using punctuation properly.

2.2 Closely Related Prior Work

As of the moment, there is no available program online that is solely designed to check the grammar of Arabic students. Most of the programs available are for all users in general, regardless if they have a different first language or if English is already their first language. One popular program designed to check the grammar in writing is the *Instant Grammarly Checker*. It has its own website and users can download their application for free. Since it is designed for all users in general and also caters to the native speakers of English, it has more advanced program that can check more than 250 grammar rules and can spot the wrong spelling used depending on the context of the sentence. Other than this, there are no other programs available online that is specifically designed to help Arab students correct their writing using the English language.

The key similarities between Instant *Grammarly Checker* and our software include that they both attempt to identify grammatical mistakes in users' sentences; they both look for more than one kind of mistake at the same time, and they both give a suggested replacement sentence to illustrate the corrections needed.

On the other hand, *Instant Grammarly Checker* was designed to help any non-native English speaker or even an English speaker with low knowledge of the grammar rules. It is also not giving instructions to users on why the suggested sentence wrote in that way. My new software designed to correct the most typical mistakes done by Arabs, and to be more educational and attractive program, the software gives the student grammatical rules related to their mistakes in the sentence they wrote if there was any. The *Instant Grammarly Checker* covered to many grammatical error in general compared to software created in this thesis.

Published research related to our project includes the grammar checking system *KNGED,* which has been designed to diagnose grammatical errors in Chinese sentences (CHANG). The approach used in this tool is based on a set of rules to identify common grammatical errors. The research included a study to analyze Chinese sentences and identify syntax errors, after which they developed an algorithm that is able to detect errors automatically and inform the student about the error type. This system is very similar in high level goals and approach to the one made in this thesis. The key difference is that *KNGED* is designed to correct the mistakes of non-Chinese language speakers while the system in this thesis would be used to correct English mistakes done by Arabian students.

Another previously published project is *CALI* (Computer-Assisted Language Instruction), which is a system that was built to help English language learners by detecting the misused grammatical rules in their sentences, hypothesizing the cause, and providing corrective information to the student. The research done for this system is similar to the work involved in creating the software of this thesis. As in our project, the research done in *CALI* is based on the assumption that the sentences provided by language learners differ in systematic ways from that of the native English speaker which is the same as saying that Arabic speakers are producing English. The key difference is that the CALI system did not attempt to engage the user by addressing learning and thus might not be as effective at preventing future mistakes.

In the following, you'll find the design work done in the new software including Requirements Specifications

Chapter 3

Design

3. Design

This chapter will present the process used to develop a design for the system. It will also explain the design itself. Figure 1 illustrates the general approach to developing the design. The steps toward solving the problem started by analyzing the problem to get the requirements. Then, a conceptual model sketched the visual interface of the software. After that, an executable system implemented and evaluated.

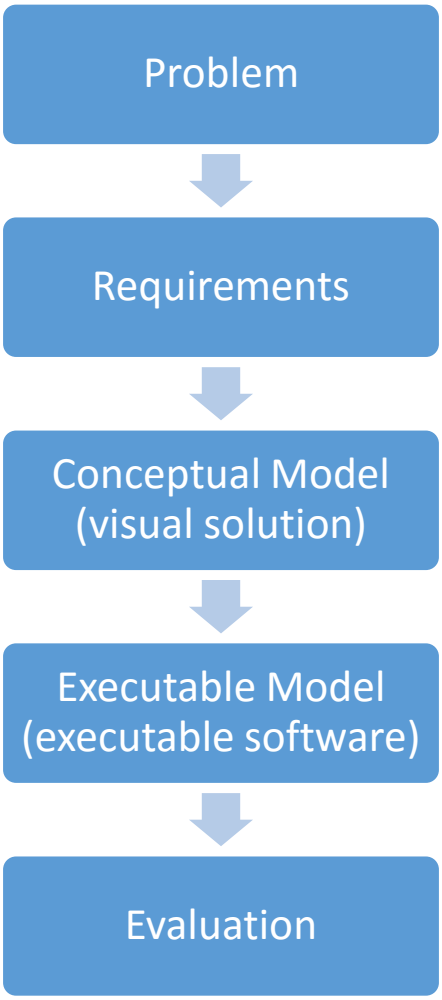

Figure 1: Theory towards solving the problem.

In what follows, we will discuss the approach in greater detail.

3.1 Problem

The problem, as has been specified, is that Arabic students make typical mistakes in writing English sentences that make the passing of college courses difficult for them while studying in English. The typical mistakes are as follows:

1- Arabic does not use capitalization and so students tend to avoid upper case in English as well. For example, students are more likely to write the first letter of the first word in the sentence with a lower case letter. They also sometimes make the same mistake with the first letter of the cities, persons, countries, and all the proper nouns but not as the first letter in the first word of the sentence.

2- Arabic starts the writing from right to left, which is the opposite of the English writing, so students may reverse the ordering of the adj-noun in their English sentences. For example: Arabs might write “I have an ID state” instead of “I have a state ID”.

3- Arabic has no verb to be or other auxiliary verbs. It is also make distinctions between the singular and plural nouns through morphology in a way that makes the verb look different for the singular name compared to the verb of the plural noun. Thus students commonly omit auxiliary verbs or fail to use the proper form of the verb *to be* in between the words of the English sentence.

4- Since Arabic lacks indefinite articles such as “a” and “an”, students may omit them or choose the wrong one.

5- Generally, Arabic is much looser in using punctuation than in English.

3.2 Requirements

Identifying the most typical mistakes of Arab students in their English writings raised the need to create a software that supports the following requirements:

1. A GUI (Graphical User Interface) that is easy to use.
2. The software should check for the following errors and alert the user:
    a. The first letter of the first word in every new sentence should be capitalized.
    b. Adjectives should be at the correct position compared to their nouns.
    c. Verbs (verb to be, and auxiliary verbs) and gerunds should appear at the correct positions in the sentence if needed.
    d. Punctuation should appear at the correct position including full-stops, commas, and question marks.
    e. The use of an – a should appear correctly in the sentence if needed.
    f. Possessive nouns should appear in correct ordering within the sentence.
3. A written sentence for the user is needed to represent the suggestion of what the correct sentence should looks like. This suggestion appears as a response of checking the user written sentence.
4. Show the user the grammatical rules they violated to let users learn from their mistakes and to make the software more a tool to support learning than as a critic that simply checks and corrects mistakes.

5. Provide the user with a help box that tells about the system, the author, and contact method for improvement purpose.

3.3 Conceptual Model

To achieve the requirements provided above, the author developed the following simulated interface (see figure 2).

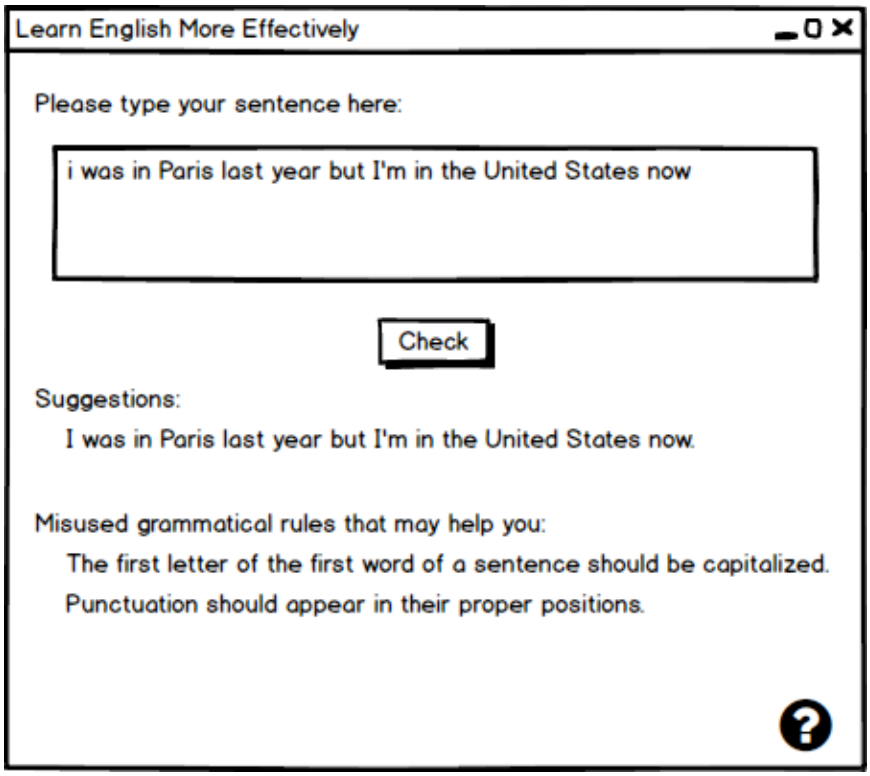

Figure 2: User interface design.

The above interface is intended to be a sketch to be used later in the implementation step. It contains a text box where the user should type the sentence, a check button to be pressed after typing the sentence for checking purpose, a suggestions section where the suggested correction for the sentence should appear if there was any mistake, the grammatical

rules sections where the student can learn from the mistakes he/she made, and a question symbol that pop ups information about the software and the creator. Note that both suggestions and grammatical rules sections should result in the response “It looks good, but please double check that you used the correct tense throughout all the sentence.” This comment was chosen by the author because the software will not address all the tense grammatical mistakes. Instead, the program will initially be more focused on the present tense. The previous figure (figure 2) was drawn using Balsamiq Mockups 3.

3.4 Executable Model

For the designed previous interface to be functioning appropriately, all the requirements should be achieved using programmed functions intended to work in each requirement. At the end of programming such functions, an executable software that looks like the designed widow should detect, correct, and instruct the Arabic user in the first place to learn from their mistakes. The next chapter will discuss the implementation step in more details.

3.5 Evaluation

The work was evaluated in several stages. First, a pilot usability test was done for this software by the author. Then the author conducted an observation study that involved22 Arabic students who used the study while being observed by the author. Lastly, a written usability and usefulness survey was completed by the same 22 subjects to measure their degree of satisfaction, to learn what they liked in the software, and what improvements they might suggest.

Those evaluation measures, along with the results and the analysis of them, will be discussed later in more detail in the evaluation chapter.

Chapter 4

Implementation

4. Implementation

Linux was used as the primary development platform for creating the software, while testing and running the code was done using Windows. We used Python 2.7 for the implementation. Python was chosen for this work because it includes ready to use library packages that can analyze the natural language, including *NLTK* (the Natural Language Toolkit). These library packages are able to classify, tokenize, stem, tag, and parse English words. Moreover, for the creation of the GUI, Python has a *TKInter* library packages, which were helpful in designing the software interface.

In the previous -design- chapter, the author converted the requirements to a conceptual model which captures the look of the GUI that the user should see and use. In this chapter, we consider the executable model in detail.

4.1 High Level Scenario of the Software

As the next diagram shows, after the user provides a text and presses the “Check” button, it initiates the system's role in helping them improve the grammar of the text. Specifically, the system interacts to do the following: (1) detect any and all typical mistakes mentioned in the design chapter, (2) print a response indicating either no correction found in case no errors were detected or print the corrected version of the sentence in case there is a mistake, (3) In case of having mistakes in the sentence, the system will also print any known grammatical rules related to the mistake(s) occurring in the user sentence. The third interaction is intended for the purpose of allowing the user to learn to avoid similar mistakes in the future.

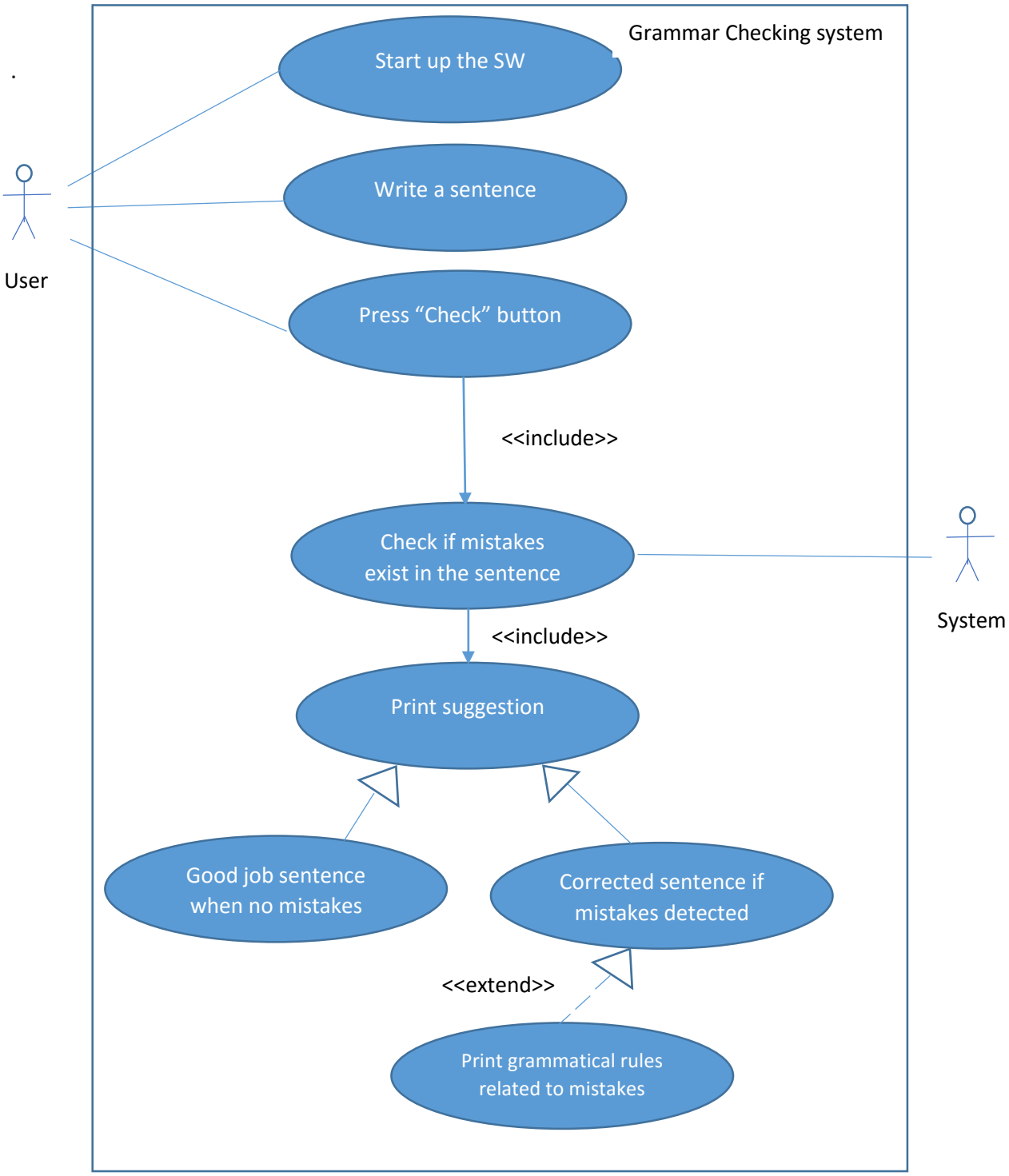

Figure 3: Use case diagram.

4.2 General Architecture

The requirements specified previously were converted to a technical requirement to help implement the code. To make this transformation, the author specified each of the technical requirements as the names of functions where each one of them is a solution to one or more of the analyzed requirements.

The functions used in this software to achieve the required goals are as follows:

1- “tagsToString”: To convert the sentence tags to string.

2- “Error1”: To detect if noun is preceding an adjective.

3- “Error2A”: To detect verbs and if gerund is used without to be.

4- “Error2B”: To find whether an extraneous modal is used with ‘do'.

5- “Error3A”: To add punctuation to the sentence where it is missing, including commas and full-stops.

6- “Error3B”: To capitalize the first letter in words at the beginning of the sentence.

7- “Error3C”: To correct the use of articles “a” and “an”.

8- “Error4”: To correct genitive construction errors.

9- “checkErrors”: The main function to check all of the errors in the provided sentence by the user.

10- “analyse”: The function to analyze and return the suggestion as per the grammatical error rule detected.

11- “Centre”: The function to provide the graphical user interface.

12- “helpMsg”: The function to show up the help message box.

4.3 Low Level Scenario of the Software (Programming Work)

Now, as we have the general overview of the software functionality, we will go much deeper to describe the tasks that each function does.

In the very beginning, the system will attempt to recognize the sentence words given to it by the user. The words then will be tagged with its matching part of speech in the language using *NLTK* library packages.

The function “tagsToString” is used to convert the tags given to each word into a string form to make it possible for the system to deal with the words in the provided sentence. “Error1” in the code detects the nouns and their adjectives. Having those two kind of words in the sentence based on the tags given to each word is used in correcting the mistakes related to the adj-noun ordering.

The rest of the functions- “Error2A”, “Error2B”, “Error3A”, “Error3B”, “Error3C”,” Error4”- are also applied to address different typical mistakes in English writing made by Arab students, as stated in the design chapter. All of the tasks rely on the tags of the words; the specific details, including the Python code that was used, are given in the appendix.

The main function that controls the rest of the error detection functions is called “checkErrors”. This function saves the mistakes that occurred in the sentence and send the results to the “analyse” function, which makes the corrections and prints the suggested correction. In case no mistakes were detected, the latter function returns “It looks good as long as your sentence is in the present tense. Good job.” This message notifies the user that no mistakes were found as long as the sentence is in the present tense since the program is limited to handling only the present tense correctly at this time.

The last two function - “Centre” and “helpMsg”- uses the *TKInter* library packages. “Centre” is to provide the graphical user interface to the student or the user and “helpMsg” is to pop-up the help window where the user can see brief information about the software.

4.4 Running the Software

To run this software, there are three Pre-requisites:

1- Python 2.7 or above should be installed in the system.

2- NLTK (Natural Language Toolkit) library packages should be installed.

3- TKInter library packages should be installed.

Note that Python is platform independent which make the implementation usable on any kind of operating system. The author created the code on a machine running Linux and ran it using both Linux and Windows. For the python file to be executed, one must provide the following commands in the *Command Prompt:*

1- For Linux:

- chmod +x Grammar_Check.py
- ./Grammar_Check.py

2- For Windows:

- cd c:\Python27 (or to the directory where you Python is installed)
- python.exe Grammar_Check.py (or the name of your python file)

After having these steps, the software should start by showing the user interface. The figure below (figure 4) represents the start-up window. The user will see that he/she needs to

type a sentence in the text box as indicated by a request above it asking the user to do that. After typing the sentence, user can then press the “Check” button to let the system starts working on detecting any grammatical errors.

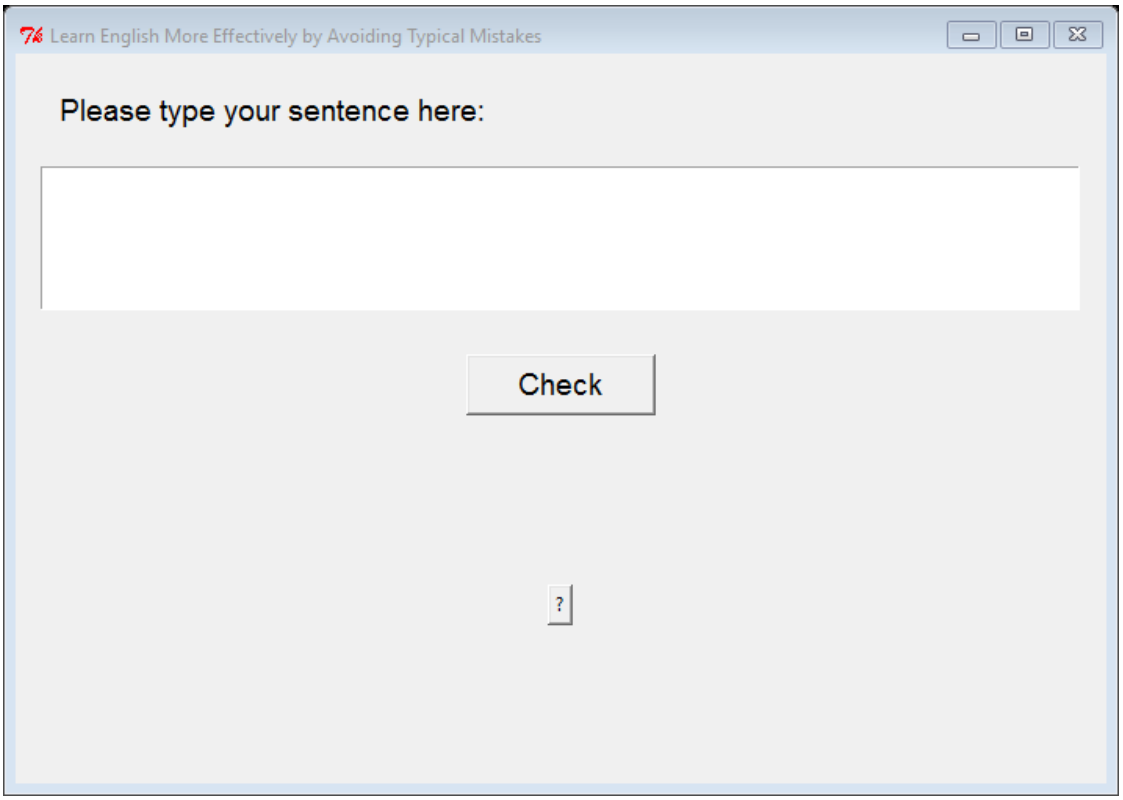

Figure 4: Software start-up window.

In the following, we consider examples of each typical mistake mentioned in the design chapter

4.5 Examples

We start with the first typical mistake, which involves capitalization. The relevant rule states that “The first letter in the beginning of every new sentence should be capitalized”. The

screenshot in figure 5 shows how the system properly corrected the lower case letter “i” in the first word “it”. Also, the system successfully provides the related grammatical rule for the user that states why the sentence was corrected in this way.

.

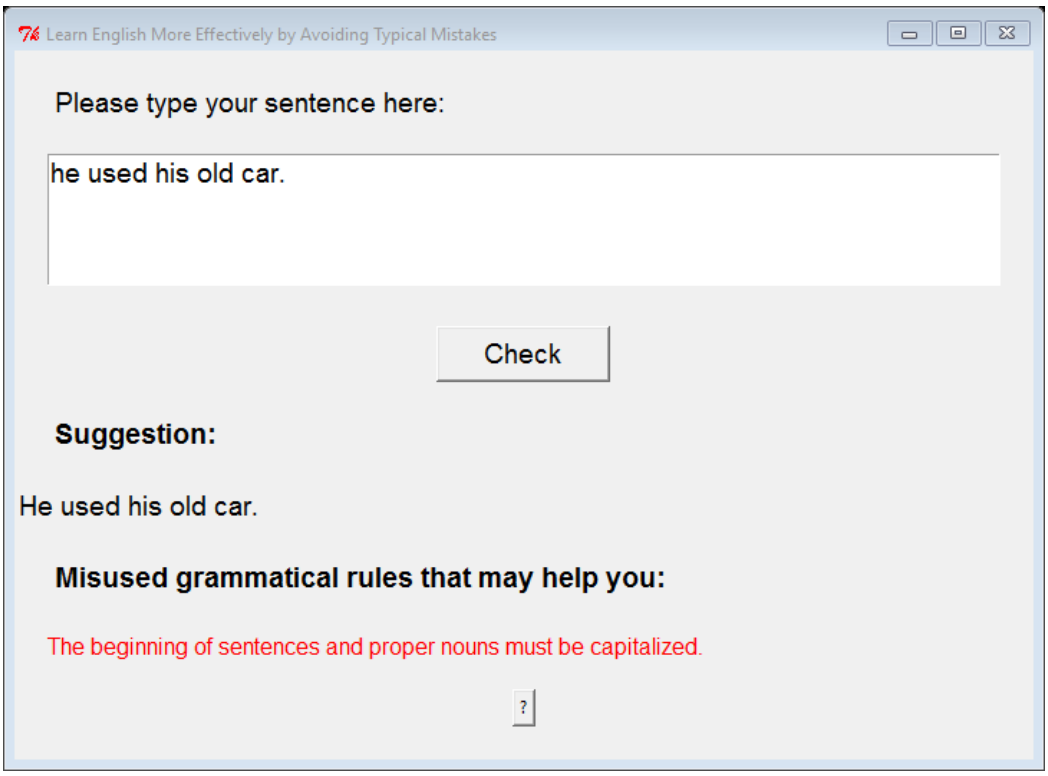

Figure 5: testing capitalization of first letter in the sentence.

The next figures illustrate the rest of the typical mistakes. Please refer to the figure description under each one for more clarification.

Note that the examples are sentences written randomly. Each sentence represents a typical mistake category.

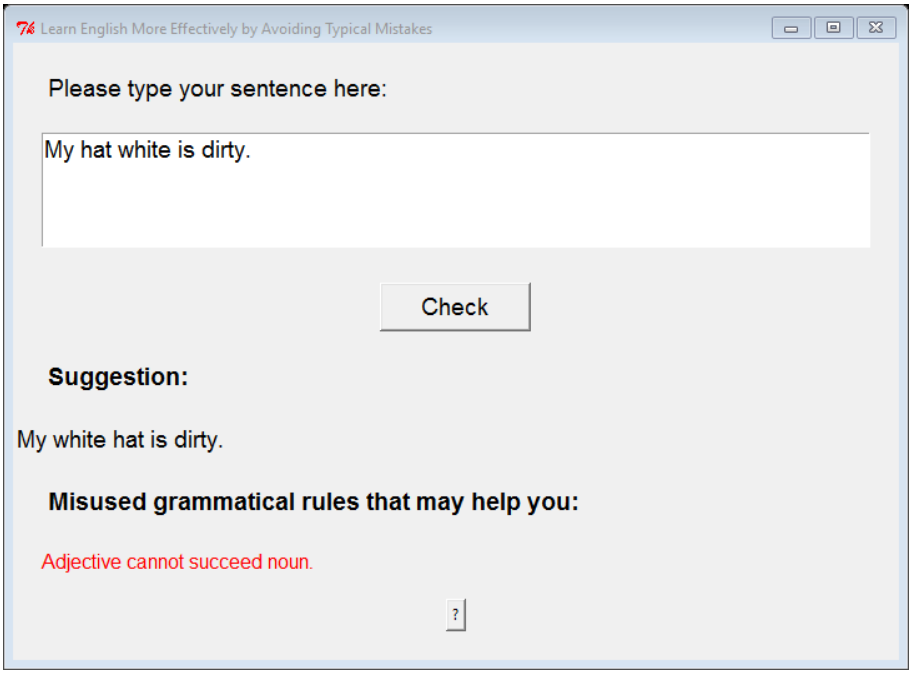

Figure 6: Testing noun-adj ordering in the sentence.

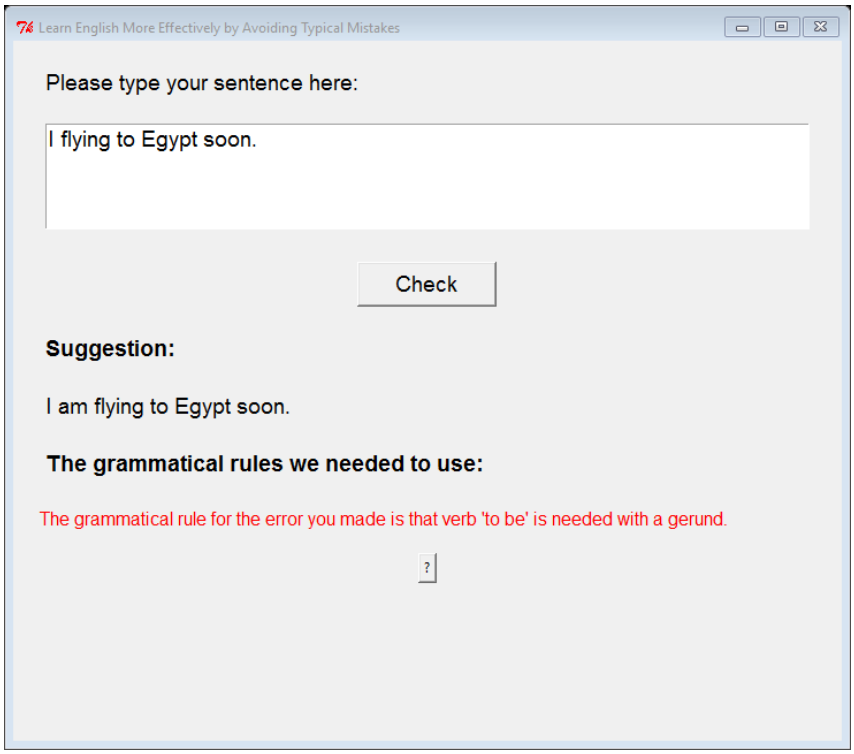

Figure 7: Testing the need of verb "to be" with gerunds.

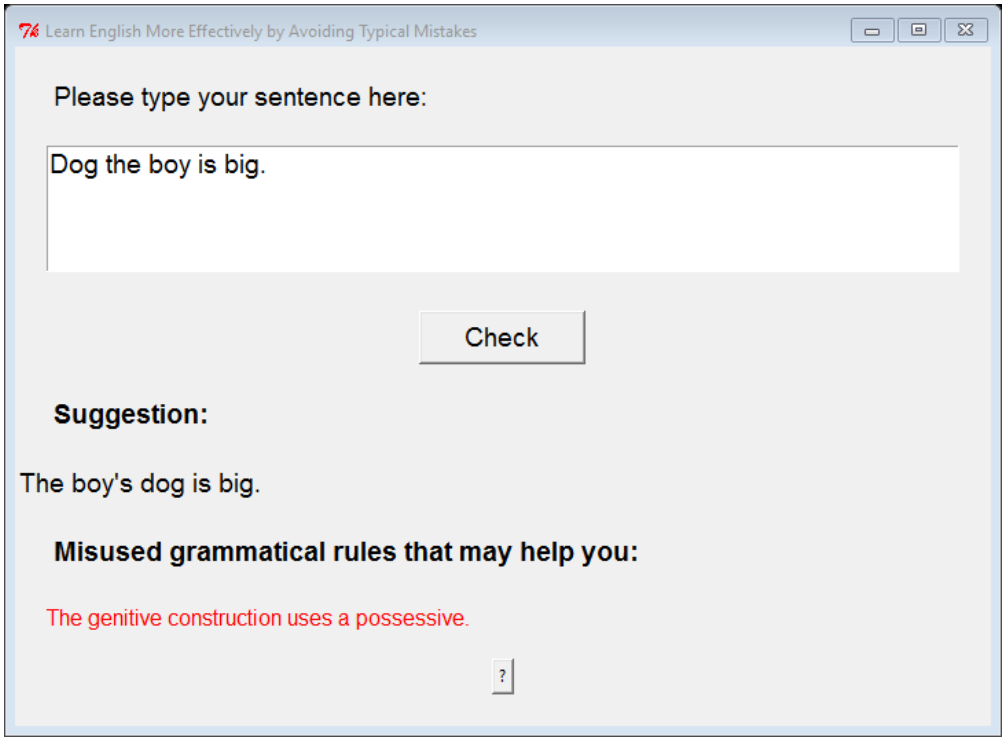

Figure 8: Testing genitive construction errors.

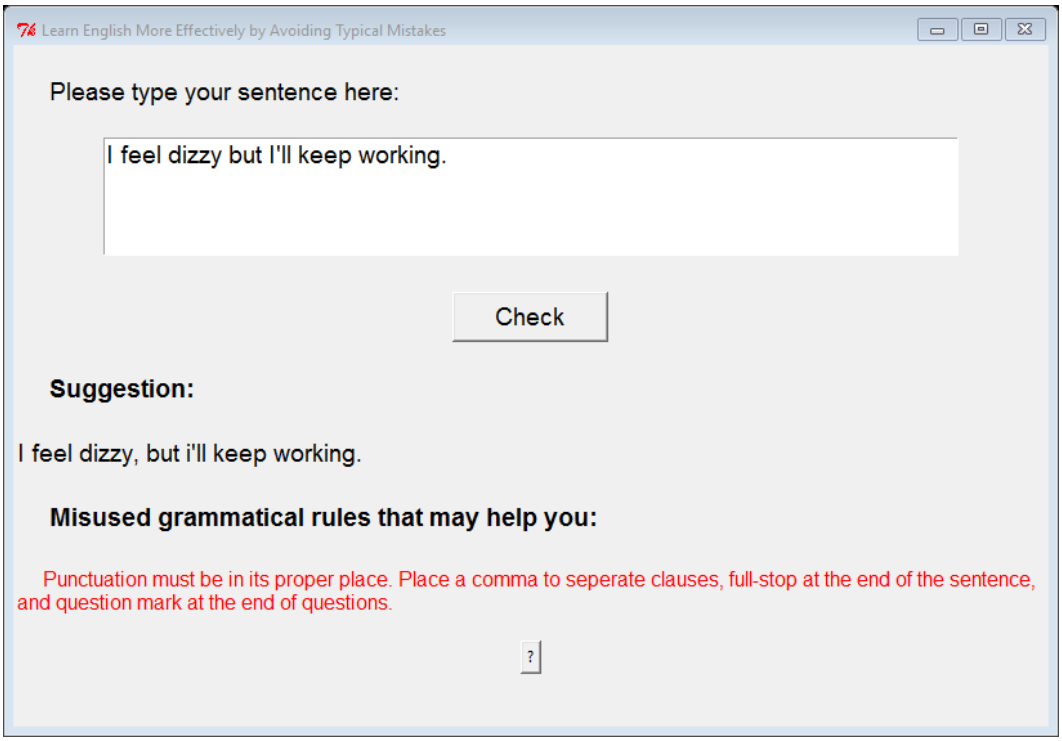

Figure 9: Testing punctuation placing errors.

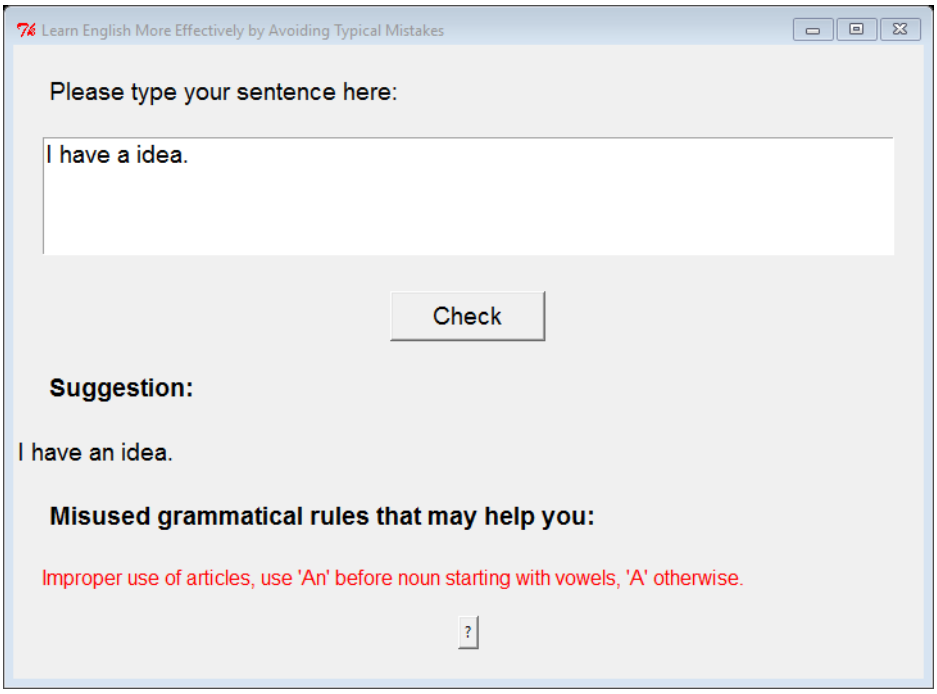

Figure 10: Testing "a" and "an" errors.

Those are all the typical mistakes illustrated with different examples. Of course there remain some limitations related to the use of tenses other than the present. This limitation, and any others, should be considered later as future work.

Lastly, it always important to let the user able to reach a help within the software application. For this reason, the author provided a help button where the student can request to see a message box with brief information about the software along with the contact email. This characteristic is beneficial also for the author. It makes it easy to contact the users to help them and learn more about their problems, comments, and suggestions about the work. Thus, the author hopes to be able to improve the work later. See figure 11 for the appearance of the help textbox message.

.

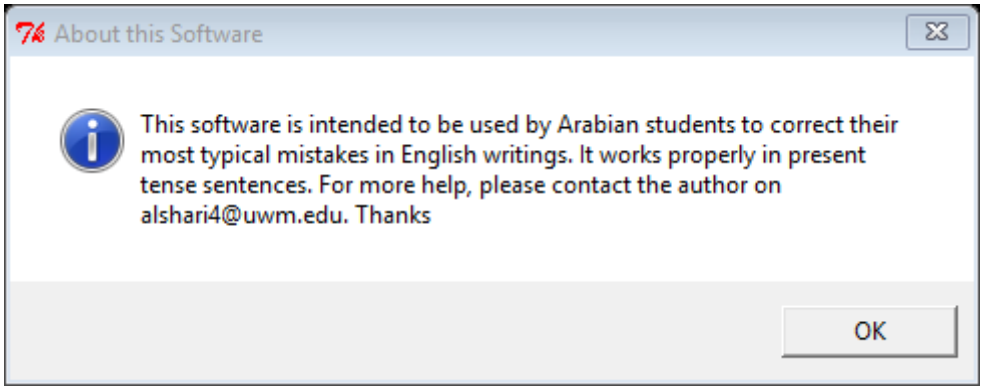

Figure 11: Help message box window.

Next, software evaluation will be discussed in detail.

Chapter 5

Evaluation

5. Evaluation

5.1 Instruments Used for Evaluation

To evaluate the software, the author used the results of two methods. An observation and a survey. For both methods, the subjects were Arabian students because the software was intended to help this population. Multiple methods were used because sometimes, people are unable to predict or explain the problems the experience while using software using a survey, however, makes it easier to compare results across subjects.

In the observation study, 22 Arabian students were recruited and asked to use the software while the author noted any problems of their using it.

Afterwards, a survey was used to ask the subjects to answer items intended to assess their perceptions of the effectiveness of created software. The survey was called “Usability and Usefulness Survey” (See Appendix A). Using this tool, the author asked the subjects, after using the software, about the most liked typed of grammatical errors which the software helped them find and fix. It included an open-ended question, so the respondents were free to give their complete and honest answers.

The survey also included four questions using a Likert Scale from 1 to 5 with 5 being the most positive answer while 1 being the most negative. The four questions were: (1) How comfortable were you with the visible presentation of the documents in the interfaces? (2) How easy was it for you to use the system? (3) How satisfied were you with the speed of the system in providing you results? (4) How satisfied were you with the feedback the system provided to

help you improve your writing? At the end of the survey, subjects were asked for any suggestions they could give to improve the usability and usefulness of the software created by the author.

5.2 Results

There were 22 Arabian students who completed the study. Subjects had different answers in the first open-ended question. The most frequently cited preferred feature was mentioned by about 23% (N=5) of participants Five of the respondents reported that they liked the software in correcting the genitive construction of the sentence. Four of the respondents expressed that they liked the system in fixing capitalization errors. Another four Arabian respondents said that they liked the software when it corrects the placing of punctuation throughout the sentence. The same number of students (four) liked detecting and correcting the “a” and “an” errors in their sentences. Meanwhile, two of the students who answered the survey like the software in detecting when they should put a question mark (“?”) for interrogative sentences. Another two students liked the software in correcting the punctuation appearance mistakes within the sentence. Meanwhile, only one student liked having a system that is able to correct the ordering of adjectives-noun sequences in the sentence.

5.3 Discussion of Evaluation

Although many students had different answers, their answers can be grouped into just five answers that include the following:

- Punctuation placement (8 respondents)
- Genitive construction (5 respondents)
- Capitalization errors (4 respondents)
- “a” and “an” errors (4 respondents)
- Adjective-noun ordering (1 respondent)

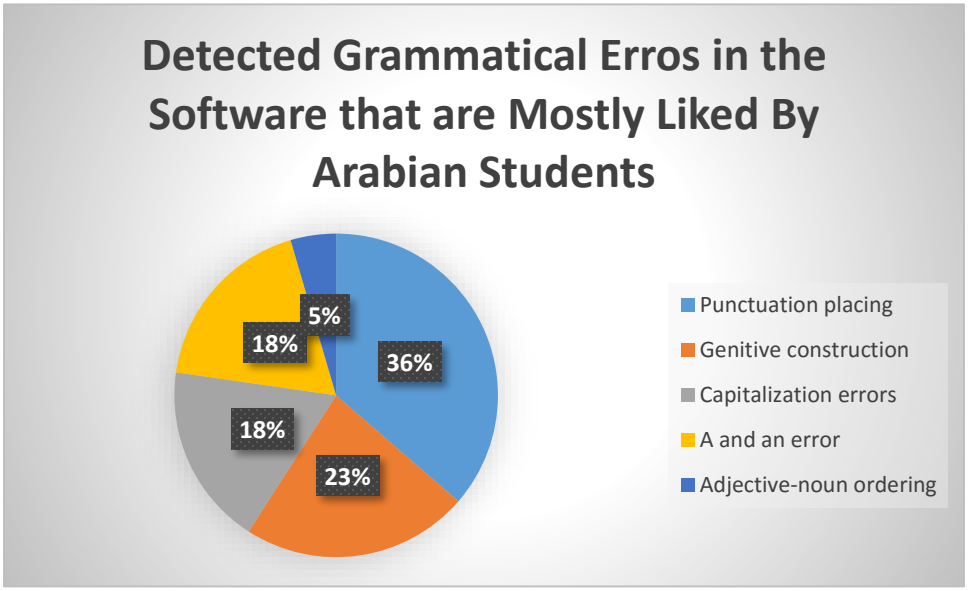

Figure 12: Detected Grammatical Errors in the software that are Mostly Liked by Arabian Students.

Figure 12 above shows exactly the percentage of each error category reported as useful in the answers of the respondents. This shows that 36%of the respondents liked how the software fix punctuation problems. In other words, just over one third of the Arabian students wanted a program that is able to put the missing punctuation in their English sentences, replace the wrong punctuation with the correct one, and remove punctuation that are misplaced in the

sentence. This also includes a program that will put a period, question mark, or any other appropriate punctuation mark in the sentence depending on the kind of sentence.

In addition, 23% of the 22 Arabian students who participated in the survey liked the software fixing the genitive construction of two nouns in a sentence. Since Arabic language has a different way of structuring nouns to show possession, this is a common mistake that most Arabian students make once they start writing in English since they may base the way they write on how the words are structured in the Arabic language. Therefore, it is not surprising that most students wanted a program that corrects this kind of typical mistake.

Meanwhile, both capitalization errors and "a"- "an" errors were mentioned by 18% each. This shows that a number of Arab students are challenged in using capital letters and "a"- "an" in the sentence. This is also understandable because the Arabic language does not have lower case or upper case in its writing system. Moreover, "*a"* and "*an"* or its equivalent are not present in the Arabic language. It appears that, as expected, most Arab students would be happier if they found a software that checks and fixes the use of *"a"* and "*an"* in their English sentence.

Furthermore, only 5%, or the smallest percentage of respondents, expressed that he/she liked he software detecting the errors in the order of the adjective-noun pairs within their English sentences. In the Arabic language, the adjective always comes after noun which is the opposite in English language where the adjective comes before the noun it modifies. This difference between the rules in grammar of the two languages may create confusion to the Arabian students who are just beginning to learn English. Also, even those who are used to writing English letters for a while, may make mistakes because of the opposite nature of the Arabic language.

Thus, it is understandable that some Arabian students would want a program that would correct the ordering of adjectives and nouns in the sentence.

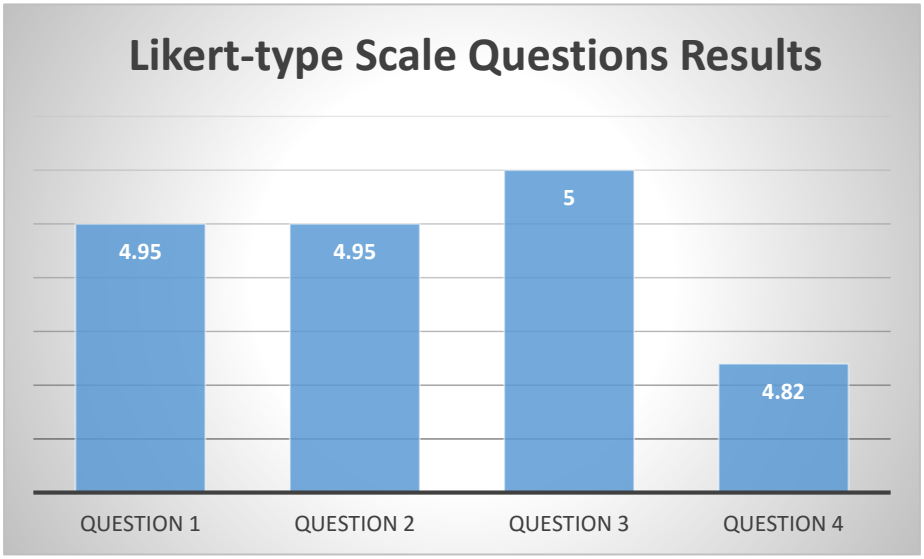

Figure 13: Likert Scaling Questions Results.

Moreover, the feedback of the survey participants about the program in general was positive. Figure 13 above shows the results of the Likert Scaling questions with a scale of 1-5, with 5 being the most positive and 1 being the most negative answer. The third question had an average score of 5 which means that all the respondents thought that the speed of the system of providing results was fast. Meanwhile, the first and the second question got the same average score of 4.95 which means that almost all the respondents became comfortable with the visible presentation of the documents in the interface. Aside from that, they also thought that it was easy to use the software. The fourth question had the lowest score of 4.82, although this score still means that majority of the students were satisfied with the feedback given by the system to help them improve their writing.

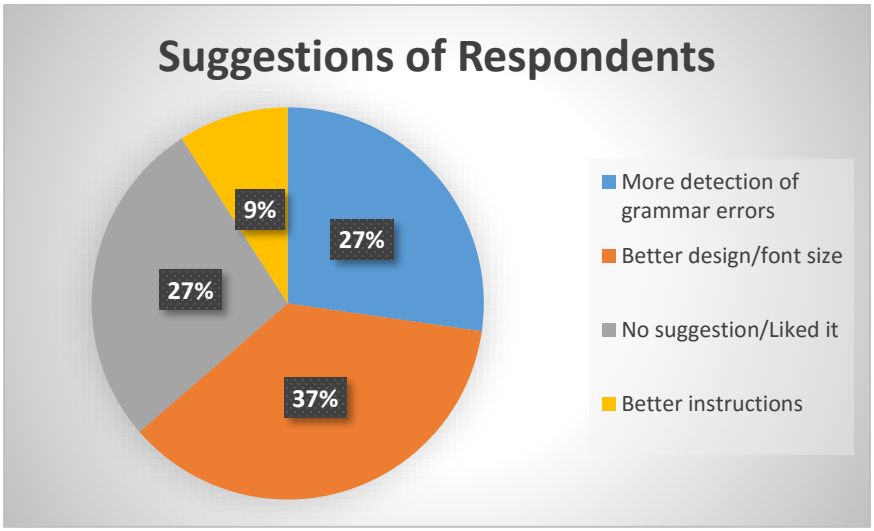

Figure 14: Suggestions of Respondents.

The last open-ended question asked users for suggestion to improve the system. We were most interested in this question because it can help one to improve the software. As seen in Figure 14 above, most of the respondents said that they would like the interface to be more attractive and that includes having a larger font size and more attractive color or attractive design in general. This represents 37% of those 22 Arabian students who used the program. In addition, some of them mentioned that the system should be more capable of detecting more punctuation errors. This comprises 27% of the respondents who reported that they want the program to detect all grammar errors if possible. Meanwhile, the same number of respondents (or another 27% of them) said that they have no suggestions for the improvement of the system and they already like it the way it is. For the author, this doesn't mean that they completely like the software, but perhaps they could not provide any suggestions for improvement at the time of participation.

Since most of the concerns were related to the attractiveness of the interface, these respondents must have been aware that the main purpose of the program is to correct just the

typical mistakes that Arabian students commit in writing sentences in English. Furthermore, the smallest portion of the respondents –only %9 or 2 students- mentioned the instructions or that the suggested answers should be clearer.

In the next and final chapter, you'll find a summary what the author did so far regarding this software, why it was successful, and some recommended directions for future work.

Chapter 6

Conclusion

6. Conclusion

6.1 Summary

For this thesis, the author chose to focus on the problems that Arabian students face when writing in English. This problem was chosen because the author is related to it directly since she is an Arabic student. To identify the problem, the author began with her informal knowledge of the most frequent mistakes made by Arabian students around her during English classes and when she was giving them her opinion about the essays they were writing. Also, she found confirmation of the occurrence of the same error types in published papers regarding the most typical mistakes of Arabian students.

For this thesis, the author worked towards finding a solution or an idea of what a possible successful solution to the problem would be. She found out that creating software to help Arabic speakers improve their English writing would be likely to be useful, if well-accepted. To enhance acceptability, she designed the software as more a learning tool than just one to detect and fix the mistakes for the users.

The software was created to satisfy many requirements identified by the author after a careful analysis of the problem. The requirements included detecting and correcting different types of English grammatical writing mistakes. They also included the need to notify the users of the specific grammatical rules that they need to learn to improve their writing, based on on the mistakes they made.

The software was tested by 22 Arabian students. To evaluate the software, the author observed the participants using the software. She looked for any unanticipated difficulties as well

as recording occurrences of expected ones. Subjects were also given a chance to report their own perceptions, through a written usability and usefulness survey at the end of using the software by the same 22 Arabian students. The answers to the questions provided in the survey suggest that most students like these would like how the software treats the punctuation mistakes and the errors related to the genitive construction. The results also revealed that students were satisfied generally with the use of the software and most of the suggestions recommend improving the efficiency of the software and the interface design, such as using a larger font size.

6.2 The Importance of the Work

For those students whose first language is not English, such as Arabian students, having a work that is intended and created to help them would be valuable. Among all what had been created to help Arabian students so far to improve their English writings, there is no such work regarding its idea. Note that there are some available programs online that helps Arabian students learn English grammar for writing. Some of them include complete sets of lessons from beginner level to advanced level. However, such programs either looks like the learning courses which are available already for Arabian students in their daily classes, or they are more grammar checking only without having the learning aspect to them. Thus, combining all of these characteristics, including checking, fixing, and teaching, along with detecting the most typical mistakes that Arabian students make is the primary contribution of the work.

The work appears to be largely successful, as 27% of the participants of the usability and usefulness survey said that they found the software to be good enough and needing no further improvement.

6.3 Limitations and Future Work

A few limitations of this work were revealed by the results of the evaluation done for the software. The most important concern is that the software user interface needs to be more attractive for the students to be more satisfied. The second most important limitation is the need to expand the number of grammatical errors that the software can detect, fix, and provide grammatical rules related to them. Arabian students have a broad range of typical mistakes and the time limitations and the difficulty of fixing the mistakes, caused the author to focus on just a few of them to start.

Moreover, to design a system that can solve all the typical mistake of Arabian students, one needs the help of an expert (or native speaker) of English because there are mistakes that a non-native speaker might not be able to identify. For example, a non-expert might write a sentence like “From the possible that I’m pregnant” instead of saying “I might be pregnant”, which at the local level might look grammatically correct, but overall is not how a native speaker would write with any normal English checker, even the ones embedded ones in commercial programs such as *Microsoft Word*, this error would be missed. However, a tool targeted to second language students might be created to identify and correct errors like this, with more sophisticated methods.

Based on these noted limitations, the future work recommended by the author would be to go further in increasing the number of grammatical errors that the software can detect, solve, and provide good learning messages. Additionally, the interface of the program should be improved so that it works better for the eye and makes the use of the software more enjoyable.

Another possible improvement would be to include a spelling checker.

Lastly, the way of presenting the grammatical rules to the students might be improved, as this was mentioned by 9% of the survey participants.

References

Al-Buainain, Haifa. “Researching Types and Causes of Errors in Arabic Speakers’ Writings.” *Qatar University.* 2006.

Alsaawi, Ali. “Spelling Errors Made by Arab Learners of English.” *International Journal of Linguistics* 7.5(2015):55-67.

Catt, M., & Hirst, G. (1990). An intelligent CALI system for grammatical error diagnosis. *Computer Assisted Language Learning*, *3*(1), 3-26.

CHANG, Tao-Hsing, Yao-Ting SUNG, and Jia-Fei HONG. "Automatically Detecting Syntactic Errors in Sentences Written by Learners of Chinese as a Foreign Language."

Gomaa, Lina. “5 Writing Trouble Spots for ESL Students of Arabic.” Teaching Community. n.d. Web. < http://teaching.monster.com/benefits/articles/10068-5-writing-trouble-spots-for-esl-students-of-arabic>.

Hourani, Taiseer Mohammed Y. “An Analysis of the Common Grammatical Errors in the English Writing Made by the 3rd Secondary Male Students in the Eastern Coast of the UAE.” *British University in Dubai*. 2008 June. Web. <http://bspace.buid.ac.ae/bitstream/1234/225/1/20050055.pdf>.

"Natural Language Toolkit." NLTK 3.0 Documentation. NLTK Project. Web. 26 Nov. 2015.

“Number of Words in the English Language.” *Global Language Monitor*. 1 Jan. 2014. Web. < http://www.languagemonitor.com/number-of-words/number-of-words-in-the-english-language-1008879/>.

“The Differences between English and Arabic.” *Frankfurt International School*. n.d. Web.

http://esl.fis.edu/grammar/langdiff/arabic.htm

Appendix A:

Usability and Usefulness Survey

What types of grammatical errors were you most interested in having a system help you find and fix?

Please rate your answers to the following questions using a scale (where 1 is lowest and 5 is highest):

1. How comfortable were you with the visible presentation of the documents? in the interfaces? (1= not at all; 5 = completely comfortable)
2. How easy was it for you to use the system? (1 = not easy; 5 = very easy)
3. How satisfied were you with the speed of the system in providing you results? (1 = not happy; 5= extremely happy)
4. How satisfied were you with the feedback the system provided to help you improve your writing? (1= not at all; 5 = completely satisfied)

In the space provided, please provide suggestions to improve the usability and usefulness of the software.

Thank you,

Appendix B:

Python Functions Used in Software Implementation

```
#!/usr/bin/env python

#Import tkinter for GUI libraries
import Tkinter
from Tkinter import *
import string

#Import tkMessageBox for information and help message box
import tkMessageBox
def helpMsg():
  tkMessageBox.showinfo("About this Software", "This software is intended to be used by Arabian students to correct their most typical mistakes in English writings. It works properly in present tense sentences. For more help, please contact the author on alshari4@uwm.edu. Thanks.")

#Import nltk for language processing
from nltk import *

#Check to see if relevant text exists
try:
   sentence = "example sentence"
   tokens = word_tokenize(sentence)
#If it doesn't, download it
except LookupError:
   print("Downloading requisite English language processing code...")
   download('book')

#Convert a list of tuples (word, part of speech) to a sentence String
def tagsToString(tags):
   result = ""
   for x in tags:
      word = x[0]
      pos = x[1]
      if pos == '.' or word[0]=="'":
         result+=word
      else:
         result+=" "+word
   result = result.strip()
   result = "     "+result
   return result

#Process text to detect if a noun is found preceding an adjective:
def Error1(tags):
   error = False
   if len(tags) < 2:
      return error
```

```
    adj = ['JJ']
    nouns = ['NN','NNS','NNP','NNPS','PRP','RP','DT']
    for i in range(len(tags)-1):
        currentWord = tags[i][0]
        currentPOS = tags[i][1]
        nextWord = tags[i+1][0]
        nextPOS = tags[i+1][1]
        if currentPOS in nouns and nextPOS in adj:
            tags[i] = (nextWord, nextPOS)
            tags[i+1] = (currentWord, currentPOS)
            error = True
    return error

#Process text to detect if a gerund is used without 'to be':
def Error2a(tags):
    error = False
    flag = 0
    if len(tags) < 2:
        print "Hemant 1"
        return error
    nouns = ['NN','NNS','NNP','NNPS','PRP','RP']
    verbs = ['VB','VBD','VBG','VBN','VBP','VBZ','MD']
    capitals = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    tagsLength = len(tags) - 1
    for i in range(tagsLength):
        print "Hemant 2"
        firstWord = tags[i][0]
        print "Hemant 2a"
        print firstWord
        firstPOS = tags[i][1]
        print "Hemant 2b"
        print firstPOS
        secondWord = tags[i+1][0]
        print "Hemant 2c"
        print secondWord
        secondPOS = tags[i+1][1]
        print "Hemant 2d"
        print secondPOS
        print "for is am"
        print tags[i][0]
        print tags[i+1][0]
        if tags[i][1] in ['VBD','VBG','VBP','VBZ'] and tags[i+1][1] in ['VBD','VBG','VBP','VBZ']:
            print "in if cond"
            tags.remove(tags[i+1])
            break
        if firstPOS in nouns and tags[i][0] not in capitals:
            print "Here"
            newword = tags[i][0].capitalize()
            tags.remove(tags[i])
            tags.insert(i,(newword,'NN'))
            break
        if secondPOS in ['VBD','VBG','VBP','VBZ'] and firstPOS not in verbs:
```

```
        for j in range(i,len(tags)):
            print "Hemant 2e @@@@@"
            print tags[j][0]
            print tags[j][1]
            if tags[j][0] == '.':
                if firstWord in ['I']:
                    print "Hemant 3"
                    if tags[i+1][0] not in ['Am','am','like',
'liked','can','could','had','have','will','would','love','loved','see','saw','hate','hated',
'want','wanted','need','needed','own','owned','belong','hear','heard','smell','seem','seemed','know','knew','believe
','believed','remember','remembered','doubt','doubted','dislike','disliked','understand','understood','suspect','susp
ected','loath','loathed','forget','forgot','prefer','preferred','feel']:
                        tags.insert(i+1,('am','VBP'))
                        flag = 1
                        error = True
                        break
                elif firstWord in ['He','he','She','she']:
                    print "Hemant 4"
                    if tags[i+1][0] not in ['Is','is','like',
'liked','can','could','had','have','will','would','love','loved','see','saw','hate','hated','want','wanted','need','needed','
own','owned','belong','hear','heard','smell','seem','seemed','know','knew','believe','believed','remember','rememb
ered','doubt','doubted','dislike','disliked','understand','understood','suspect','suspected','loath','loathed','forget','f
orgot','prefer','preferred', 'feel']:
                        tags.insert(i+1,('is','VBP'))
                        flag = 1
                        error = True
                        break
                elif firstWord in ['They','they','We','we','You','you']:
                    if tags[i+1][0] not in ['Are','are','like',
'liked','can','could','had','have','will','would','love','loved','see','saw','hate','hated',

'want','wanted','need','needed','own','owned','belong','hear','heard','smell','seem','seemed','know','knew','believe
','believed','remember','remembered','doubt','doubted','dislike','disliked','understand','understood','suspect','susp
ected','loath','loathed','forget','forgot','prefer','preferred','start','started','feel']:
                        tags.insert(i+1,('are','VBP'))
                        flag = 1
                        error = True
                        break
            elif tags[j][0] == '?':
                if firstWord in ['I']:
                    print "Hemant 6"
                    if tags[i+1][0] not in ['Am','am']:
                        tags.insert(i,('am','VBP'))
                        flag = 1
                        error = True
                        break
                elif firstWord in ['He','he','She','she']:
                    print "Hemant 7"
                    print tags[i+1][0]
                    if tags[i+1][0] not in ['Is','is']:
                        tags.insert(i,('is','VBP'))
                        flag = 1
                        error = True
```

```
                    break
                elif firstWord in ['They','they','We','we','You','you']:
                    if tags[i+1][0] not in ['Are','are','Will','will']:
                        tags.insert(i,('are/will','VBP'))
                        flag = 1
                        error = True
                        break
        if flag == 1:
            break
    return error
#Process text to find if an extraneous modal is used with 'do'
def Error2b(tags):
    error = False
    if len(tags) < 2:
        return error
    verbs = ['VB','VBD','VBG','VBN','VBP','VBZ','MD']
    do = ['do','Do','does','Does']
    for i in range(len(tags)-2):
        firstWord = tags[i][0]
        firstPOS = tags[i][1]
        secondWord = tags[i+1][0]
        secondPOS = tags[i+1][1]
        thirdWord = tags[i+2][0]
        thirdPOS = tags[i+2][1]
        if firstWord in do and secondPOS == 'MD':
            tags[i+1] = (('',''))
            error = True
        elif firstWord in do and thirdPOS == 'MD':
            tags[i+2] = (('',''))
            error = True
    while (('','')) in tags:
        tags.remove(('',''))
    return error
#Add punctuation to the sentence where it does not exist:
def Error3a(tags):
    error = False
    if len(tags) < 2:
        print "Hemant 3a 1"
        return error
    capitals = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    question
=["Who","What","Where","When","Why","How","who","what","where","when","why","how","Is","Are","is","are"
,"Do","do","Should","May","Could","should","may","could","would","Would","Does","do"]
    print "Questionmark condition"
    print tags[0][0]
    print tags[len(tags)-1][0]
    if tags[0][0] in question and tags[len(tags)-1][0] != '?':
        print "Add questionmark"
        i = len(tags) - 1
        tags.remove(tags[i])
        tags.append(('?','.'))
        error = True
    if tags[-1][1] != ('.'):
```

```
    for x in range(len(tags)-1,-1,-1):
        if tags[x][1] in ['.','?','!']:
            print "Hemant 3a 2"
            break
    if x != 0:
        print "Hemant 3a 3"
        x += 1
    if tags[x][0] in question:
        print "Hemant 3a 4"
        tags.append(('?','.'))
    else:
        print "Hemant 3a 5"
        tags.append(('.','.'))
    error = True
for i in range(len(tags)-1):
    print "Hemant 3a 6"
    currentWord = tags[i][0]
    currentPOS = tags[i][1]
    nextWord = tags[i+1][0]
    nextPOS = tags[i+1][1]
    if nextPOS != 'NNP' and nextWord != "I" and nextWord[0] in capitals and currentPOS != '.':
        for x in range(i,-1,-1):
            if tags[x][1] in ['.','?','!']:
                print "Hemant 3a 7"
                break
        if x != 0:
            print "Hemant 3a 8"
            x += 1
        if tags[x][0] in question:
            print "Hemant 3a 9"
            tags.insert(i+1,('?','.'))
        else:
            print "Hemant 3a 10"
            tags.insert(i+1,('.','.'))
            error = True
verbs = ['VB','VBD','VBG','VBN','VBP','VBZ','MD']
fanboys = ['for','and','nor','but','or','yet','so','because','while','although','therefore','thus']
addCommas = []
left = False
right = True
for i in range(len(tags)-1):
    print "Hemant 3a 11"
    currentWord = tags[i][0]
    currentPOS = tags[i][1]
    if currentWord in fanboys:
        for j in range(i,len(tags)):
            if tags[j][1] in verbs:
                print "Hemant 3a 12"
                right = True
                break
            elif tags[j][1] == '.':
                print "Hemant 3a 13"
                break
```

```
            for j in range(i,-1,-1):
                if tags[j][1] in verbs:
                    print "Hemant 3a 14"
                    left = True
                    break
                elif tags[j][1] == '.':
                    print "Hemant 3a 15"
                    break
            if left and right:
                print "Hemant 3a 16"
                error = True
                addCommas.append(i)
    for loc in addCommas:
        print "Hemant 3a 17"
        tags.insert(loc,(',',','))

    removeCommas = []
    for i in range(len(tags)-1):
        if tags[i][0] == ',' and tags[i+1][0] not in fanboys:
            print "Hemant 3a 18"
            removeCommas.append(i)
            error = True
    for loc in removeCommas:
        print "Hemu"
        print tags[loc]
        del tags[loc]
        error = False
        print error
        break

    nouns = ['NN','NNS','NNP','NNPS','PRP','RP']
    print error
    for i in range(len(tags)-1):
        print error
        if tags[i][0][-1] == 's' and tags[i][0][-2] != "'" and tags[i+1][1] in nouns:
            print "Hemant 3a 19"
            tags[i] = (tags[i][0][:-1]+"'"+tags[i][0][-1],tags[i][1])
            error = True
        print tags[-3][0]
        if tags[-2][0] == 'please' and tags[-1][0] == "?":
            print "Hemant 3a 20"
            tags.insert(-2,(',','.'))
            error = True
            print error
        break
    print error
    return error
#Capitalize letters in words at the beginning of a sentence:
def Error3b(tags):
    error = False
    if len(tags) < 2:
        return error
    capitals = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

```
    punctuation = ".!?"
    nouns = ['NN','NNS','NNP','NNPS','PRP','RP']
    for i in range(len(tags)-1):
        currentWord = tags[i][0]
        nextWord = tags[i+1][0]
        if (i == 0 or currentWord == 'i') and currentWord[0] not in capitals:
            newWord = currentWord.capitalize()
            tags[i] = (newWord,tags[i][1])
            error = True
        elif currentWord in punctuation and nextWord[0] not in capitals:
            newWord = nextWord.capitalize()
            tags[i+1] = (newWord,tags[i+1][1])
            error = True
    return error
#Correct usage of A/An
def Error3c(tags):
    error = False
    if len(tags) < 2:
        return error
    a = ['a','A']
    an = ['an','An']
    vowels = 'aeiouAEIOU'
    for i in range(len(tags)-1):
        currentWord = tags[i][0]
        nextWord = tags[i+1][0]
        if currentWord in a and nextWord[0] in vowels:
            newWord = an[a.index(currentWord)]
            tags[i] = (newWord,tags[i][1])
            error = True
        elif currentWord in an and nextWord[0] not in vowels:
            newWord = a[an.index(currentWord)]
            tags[i] = (newWord,tags[i][1])
            error=True
    return error
#Correct genitive construction errors:
def Error4(tags):
    print "4"
    error = False
    if len(tags) < 3:
        print "4a"
        return error
    nouns = ['NN','NNS','NNP','NNPS','PRP','RP']
    for i in range(len(tags)-2):
        print "4b"
        firstWord = tags[i][0]
        firstPOS = tags[i][1]
        secondWord = tags[i+1][0]
        secondPOS = tags[i+1][1]
        thirdWord = tags[i+2][0]
        thirdPOS = tags[i+2][1]
        if secondWord == 'the' and firstPOS in nouns and thirdPOS in nouns:
            tags[i] = (secondWord,secondPOS)
            tags[i+1] = (thirdWord,thirdPOS)
```

```
            tags[i+2] = (firstWord,firstPOS)
            tags.insert(i+2,("'s", 'POS'))
            error=True
    return error
#Master function which calls all of the error checks and which
#prints the evolving structure of the sentence for debugging
def checkErrors(text):
    errors = ""
    print("Sentence:")
    print(text)
    tokens = word_tokenize(text)
    print("Tokens:")
    print(tokens)
    tags = pos_tag(tokens)
    print("Original:")
    print(tags)
    e3a = Error3a(tags)
    print("Punctuation correction:")
    print(tags)
    e3b = Error3b(tags)
    print("Capitalization correction:")
    print(tags)
    e1 = Error1(tags)
    print("Noun preceding adjective correction:")
    print(tags)
    e2a = Error2a(tags)
    print("Omission of  verb 'to be' in gerund correction:")
    print(tags)
    e2b = Error2b(tags)
    print("Extraneous use of modal verb with 'do' correction:")
    print(tags)
    e3c = Error3c(tags)
    print("A/An correction:")
    print(tags)
    e4 = Error4(tags)
    print("Genitive construction correction:")
    print(tags)
    if e1:
        errors += "    Adjective cannot succeed noun.\n"
    if e2a:
        errors += "    Verb 'to be' is needed with a gerund.\n"
    if e2b:
        errors += "    A modal verb is not needed with 'do'.\n"
    if e3a:
        errors += "    Punctuation must be in its proper place. Place a comma to seperate clauses, full-stop at the end
of the sentence, and question mark at the end of questions.\n"
    if e3b:
        errors += "    The beginning of sentences and proper nouns must be capitalized.\n"
    if e3c:
        errors += "    Improper use of articles, use 'An' before noun starting with vowels, 'A' otherwise.\n"
    if e4:
        errors += "    The genitive construction uses a possessive.\n"
    fixedText = tagsToString(tags)
```

```
    if errors == "":
        errors = "     It looks good as long as your sentence is in the present tense. Good job."
        L6.config(fg='green')
        top.update()
    else:
        L6.config(fg='red')
    return (fixedText,errors)
#Master function which analyses text
def analyze():
    text = T1.get("1.0",END)
    fixedText, errors = checkErrors(text)
    fixedText = fixedText.split()
    for index in range(1, len(fixedText)):
      if fixedText[index] != 'I':
        if len(fixedText[index])>1 :
          if (fixedText[index][0] in string.ascii_uppercase and
              fixedText[index][1] in string.ascii_uppercase) : continue
          else :
            l = len(fixedText[index-1])
            if fixedText[index-1][l-1] != '.' :
              fixedText[index] = fixedText[index].lower()
        else :
          if  fixedText[index][0] in string.ascii_uppercase :
            l = len(fixedText[index-1])
            if fixedText[index-1][l-1] != '.' :
              fixedText[index] = fixedText[index].lower()
    fixedText = ' '.join(fixedText)
    if fixedText[0] not in string.ascii_uppercase:
      if fixedText[0] in string.ascii_lowercase:
        tmp = fixedText[:1]
        tmp = tmp.upper()
        fixedText = tmp + fixedText[1:]
    #errors = checkErrors(text)
    L3v.set("\n     Suggestion:\n")
    L4v.set(fixedText)
    L5v.set("\n     Misused grammatical rules that may help you:\n")
    L6v.set(errors)
#Helper function which centers window
def center(win):
    win.update_idletasks()
    width = win.winfo_width()
    height = win.winfo_height()
    x = (win.winfo_screenwidth() // 2) - (width // 2)
    y = (win.winfo_screenheight() // 2) - (height // 2)
    win.geometry('{}x{}+{}+{}'.format(width, height, x, y))
#User interface code
top = Tk()
top.title("Learn English More Effectively by Avoiding Typical Mistakes")
top.geometry('800x800')
L1 = Label(top, text="\n     Please type your sentence here:\n", font=("Helvetica", 14))
L1.pack(anchor=W)
T1 = tkinter.Text(top,height=4,width=60,font=("Helvetica", 14))
T1.pack()
```

```
L2 = Label(top, text="", font=("Helvetica", 14))
L2.pack()
B1 = Button(top, text="     Check     ", command=analyze,font=("Helvetica", 14))
B1.pack()
L3v = StringVar()
L3 = Label(top, textvariable=L3v, font=("Helvetica", 14, "bold"))
L3.pack(anchor=W)
L4v = StringVar()
L4 = Label(top, textvariable=L4v, font=("Helvetica", 14), justify=LEFT, wraplength=800)
L4.pack(anchor=W)
L5v = StringVar()
L5 = Label(top, textvariable=L5v, font=("Helvetica", 14, "bold"))
L5.pack(anchor=W)
L6v = StringVar()
L6 = Label(top, textvariable=L6v, fg="red", font=("Helvetica", 12), justify=LEFT, wraplength=800)
L6.pack(anchor=W)
B2 = Tkinter.Button(top, text = "?", command = helpMsg)
B2.pack()



center(top)
top.mainloop()
```