**CHAPTER 9**

**IMPLEMENTATION**

**9.1 Implementation choices**

The options for the programming language for implementation mainly included C++, Java, C# and Python.

The system needed a natural language processing subsystem at the back end. The options available were Apache Open NLP, Sharp NLP and NLTK.

The considered choices for implementing the front end were Camelot, PySide, PyObjC, QT, PyQT and wxPython.

**9.2 Reasons for choosing a particular implementation**

Python was chosen as the language for implementing the back end due to its support of various NLP libraries and UI frameworks. Also, application programming using python was considered more suitable. The Python regex library is also used in the implementation, and was found very useful.

NLTK, or Natural Language Toolkit was chosen for this purpose due to its easy-to-use interfaces with many corpora and lexical resources such as wordnet, CMU dictionary, Brown Corpus, etc. NLTK also comes with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.  NLTK is a free, open source, community-driven project available for Windows, Mac OS X, and Linux. NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike.

The front end, i.e. the UI was created using PyQT version 4.10.3PyQT allows creation of the GUI with the help of QT designer that is easy to use. The .ui file can then directly be converted to .py files by using pyuic.

**9.3 Data structures used**

The data structures used to store the following details are:

* Misused words: A list with each element being a list itself. Each element has three strings – the misused word, it’s tag, correct word
* Misused phrases in Indian English: A list with each element being a list of two strings: the first one is the incorrect phrase, second is the suggested correction.
* Comments generated: The comments are stored in a list with each element containing the three things:
  + The sentence for which the comment was generated as a string
  + The severity of the error found
  + The Comment as a string
* Floweriness:
* Tokenised sentences: The nltk.sent\_tokenize() function returns a list of tokenised sentences.
* Tokenised words: The nltk.word\_tokenize() function returns a list of tokenised words.
* Tagged words: The nltk.pos\_tag () function returns a list of tagged tuples.
* Readability:
* Statistics about the text: All the statistics about the text such as number of sentences, number of words, number of syllables, number of polysyllables and number of characters are stored in a list.
* Syllable counting: A list is used for finding the number of syllables in the text, further used to calculate the readability.

**9.4 Communication mechanisms used and their structure**

* QT Action

The QAction class provides an abstract user interface action that can be inserted into widgets. In applications many common commands can be invoked via menus, toolbar buttons, and keyboard shortcuts. Since the user expects each command to be performed in the same way, regardless of the user interface used, it is useful to represent each command as an *action*.

* QT Signal/Slot mechanism

The signal/slot mechanism has the following features:

* + A signal may be connected to many slots.
  + A signal may also be connected to another signal.
  + Signal arguments may be any Python type.
  + A slot may be connected to many signals.
  + Connections may be direct (ie. synchronous) or queued (ie. asynchronous).
  + Connections may be made across threads.
  + Signals may be disconnected.

**9.5 Pseudocode**

1. For detection and correction of misused phrases in Indian English:

For each sentence in the input data

For each element in the list of misused phrases:

If the first string exists in the sentence,

Generate a comment for the sentence.

2. To calculate obscurity:

Load the word frequency distribution from Brown Corpus.

For each word, get the frequency of the word from Brown Corpus.

If the frequency is less than 5, increment count.

Divide the total count by the number of sentences to get obscurity.

3. To calculate floweriness:

Initialise adjCount as 0.

For each sentence, tag the words in the sentence.

Initialise adj, adv to 0.

For each tagged word, check if the tag is JJ or JJR or JJS. If yes,

Increment adj

For each tagged word, check if the tag is RB or RBR or RBS. If yes,

Increment adv

Add adj and adv to the adjCount

4. Checking if a sentence ends with a preposition:

For each sentence:

Get the number of words in the sentence

Compare the tag of the last word with IN. If yes,

Return true.

5. Detecting tense inconsistency:

For each sentence:

Set isPast, isPresent and quotes to false

For each tagged tuple in the sentence:

If the first element is single or double quote,

Set quotes to true

If quotes is false:

If tag is Past tense verb:

If isPresent is true,

Return false

Else:

Set isPast to true

Else if tag is present tense verb:

If isPast is true,

Return false

Else:

Set isPresent to true.

5. Checking length of the sentence:

For each sentence:

Tokenize each sentence into words.

Count the number of tokens.

If length is more than 25, generate comment about sentence being too long.

6. Detecting misused words:

For each sentence, search if any misused word is present in the sentence.

If yes:

Check if the tag matches what it should when used in the right context.. If not:

Generate comment suggesting the correct word.

7. Calculating Readability

Flesch Reading Ease:

206.835 - 1.015 \* (total words / total sentences) - 84.6 \* (total syllables / total words)

The higher the score, the easier it is to read.

Score range Implication

90-100 11 year old student

60-70 13-15 year old students

0-30 university graduate students

Readability formulae:

Give an approximation of the readability of the text by specifying the years of formal education needed to understand the text on a first reading. Hence, the higher the score, the tougher the text is to read.

Each have their own formula considering various parameters.

Flesch-Kincaid Grade Level:

0.39 (total words / total sentences) + 11.8 (total syllables / total words) - 15.59

Lowest theoretical grade score: -3.4

Gunning-Fog Score:

0.4 \* [ (words / sentences) + 100 \* (complex words / words) ]

Complex words:

- words with 3 or more syllables.

- proper nouns not included

- common suffixes like -es, -ed, or -ing are excluded

Score of 8 for near universal understanding.

Score of <12 for a wide general audience.

Coleman-Liau Index:

0.0588L - 0.296S - 15.8

L: average number of letters per 100 words

S: average number of sentences per 100 words.

SMOG Index:

1.043 \* √ (number of polysyllables \* 30 / number of sentences) + 3.1291

polysyllables: words with 3 or more syllables

Sample size of text must be at least 30 sentences.

Automated Readability Index:

4.71 \* (characters / words) + 0.5 \* (words / sentences) - 21.43