Various Uses of a Spelling Checker Project: Practical Experiences, Teaching, and Learning

Menno M van Zaanen[1] & Gerhard B van Huyssteen[2*]
[1]*Induction of Linguistic Knowledge, Computational Linguistics, University of Tilburg, Tilburg, The Netherlands*
e-mail: mvzaanen@uvt.nl
[2]*School for Languages, Potchefstroom University for Christian Higher Education, Potchefstroom, 2531, South Africa*
**Corresponding author, e-mail: afngbvh@puknet.puk.ac.za*

Abstract

In this article we give an overview of various aspects of a project developing a spelling checker for Afrikaans. We discuss two of the main aims of the project, viz. for researchers to obtain practical experience, and to further learning of both researchers and students. This article, therefore, consists of two relatively independent parts that each describes aspects related to these two aims. The first part describes the actual spelling checker. The existing spelling checker is evaluated and, based on the results, improvements are introduced and implemented. Remaining problems are discussed and possible solutions are proposed. It is illustrated how practical experience was obtained during the evolution of the project. The second part discusses the underlying teaching and learning benefits of the project. This includes improving our knowledge of computational linguistics on Afrikaans, improving programming skills for linguists, and communication between experts in different fields. We also give an overview of how the project is used for problem-oriented and project-organised educational purposes, and how it solves certain problems related to a shortage of computational linguistic tools and resources for Afrikaans.

1. Introduction

Even though South Africa has eleven official languages, very little computational linguistic research focussed on these languages in the past. Additionally, no complete computational linguistics qualification was available in South Africa. It is only since recently that directed research projects and coherent educational programmes in computational linguistics were established in South Africa.

In accordance with this general tendency, the *Potchefstroom University for Christian Higher Education* (PUCHE) established both a training and a research programme in computational linguistics. The graduate programme, BA Language Technology, was introduced in 2002, while the research sub-programme "Language and Technology" was already instituted in 2000 within the focus area, "Languages and Literature in the South African Context". Within this research sub-programme, four divisions were identified, viz. Speech Technology, Text Technology, Corpora, and Subtitling. Different projects were consolidated or undertaken in each of these

divisions, one of which is the Afrikaans spelling checker project, on which we report in this article.[1]

As a first major project, it was decided to update and redesign the existing spelling checker for Afrikaans (henceforth called *PUKspell*), that had been developed and marketed by the IT department of the PUCHE since 1998. Several possible benefits from this research were foreseen. First, it was considered to be a relatively easy project, by means of which researchers and students could get some useful practical and hands-on experience in the development of language technology applications. It does not only provide the opportunity to work with different natural language processing techniques and processes, but also gives insight in what problems arise when pure linguists have to consider what computers can and cannot do. Related are the (communication) problems that occur when linguists from their point of view need to interact with programmers (i.e. when working together in a development team). Second, the fact that students are directly involved in the research project, was seen as an important aim, as this allows them to obtain knowledge about research in computational linguistics, but also in particular about developing a spelling checker. The third aim was that a practical product is developed that can be used and possibly commercially exploited. As a side-effect, other tools, such as a stemmer, a hyphenator, a lexicon, and other "enabling" tools and resources are developed. These tools can then be re-used in future projects. Lastly, this project would hopefully contribute to the status of the language, by providing the language users with a product that will make it easier for them to create high quality texts in Afrikaans.

This article is divided into two parts, based on the first two aims mentioned above. The first part gives an overview of the actual spelling checker. It describes the currently available spelling checker, which is compared to another available spelling checker for Afrikaans. Based on these results, the weaknesses and their possible improvements are evaluated. An overview of the new spelling checker, including the improvements, is also given.

In the second part of the article, different facets of the teaching and learning processes are described. The focus here is on how the development of the spelling checker is used in education and learning. It is shown that the project has two main advantages. On the one hand, it allows the students to get a good feeling of the problems that a computational linguist may encounter by helping directly with the actual development of the spelling checker. On the other hand, it gives the linguists in the team the opportunity to grow in their roles as computational linguists (while collaborating with experts in other fields, such as computer science), by learning about natural language processing techniques and methods, and even by learning how to do simple computer programming for text processing.

2. Practical Experience

2.1 Introduction

There are two practical reasons why the development of a spelling checker is used in this project. First, the IT department of the university had already developed a working version of a spelling checker since 1998, which has been commercially exploited successfully. This spelling checker is mainly a lookup module that compares each word to words in the lexicon.

The main problem with this approach is that the performance of the spelling checker is directly related to the comprehensiveness of the lexicon. If correct words cannot be found in the lexicon, they are considered incorrect. This problem led to the demand of a more sophisticated spelling checker.

Second, the "old" spelling checker has been used commercially, which shows that there is an actual interest in an Afrikaans spelling checker. A new and improved version of the spelling checker will (hopefully) result in a profit. This profit can again be used to support further research.

In this section, we first investigate exactly what the problematic areas of the existing spelling checker are by comparing it against another commercially available spelling checker for Afrikaans. Next, we give an overview of the existing and the improved spelling checkers and discuss the implemented improvements. Finally, this section is concluded with a brief summary.

2.2 Comparison between Two Spelling Checkers for Afrikaans

Only when we know how the existing *PUKspell* performs (also compared to other existing spelling checkers), we need to know what should be improved. We therefore compare two available spelling checkers for Afrikaans: *PUKspell* and another commercial spelling checker which we will call *Bspell*[2] (Van Huyssteen, 2002).

The approach taken here is to apply both spelling checkers to a text containing correct and incorrect words. The flags and corrections suggested by both spelling checkers are evaluated against the correct text. This evaluation is done using a set of metrics. The next section will describe which metrics are used and also discuss some of the problems that remain when evaluating spelling checkers in this way. It concludes with a discussion of the results of the evaluation phase.

2.2.1 Evaluation Metrics

There are many aspects of spelling checkers that can be evaluated. In this article, we divide the aspects into two parts: *user-friendliness* (e.g. completeness and readability of the manual, ease of use and installation, etc.), and *performance* (i.e. recall, precision, and suggestion adequacy).

User-friendliness is hard to measure. In the evaluation performed here, we found that both spelling checkers are comparable with regard to user-friendliness. Both spelling checkers

perform reasonable. Based on this, no preference for one of the two spelling checkers could be made. The remaining part of the evaluation in this article will therefore focus on more directly measurable metrics.

Metric	Method of measurement
Lexical recall	# valid words accepted / # valid words
Error recall	# invalid words flagged / # invalid words
Precision	# correctly flagged invalid words / # words flagged
Suggestion adequacy	# correct suggestions for flagged, invalid words / total # of flagged, incorrect words

Table ???1: Evaluation metrics for evaluation of spelling checkers

Table ???1 gives an overview of the more directly measurable metrics that will be used here. It is useful to define some terms here:

- *Valid words* are words that are part of the language, or which are sanctioned by the language system, in contrast to *invalid words*, which are *not* part of the lexicon or language system.
- *Flagging* a word is when the spelling checker claims a word is invalid, while *accepting* a word means treating it as valid. Accordingly, a *flag* is an indication that a word has been tagged as invalid (regardless if the word really was invalid or not).
- *Suggestions* are alternative valid words that are offered to the user to replace a flagged word with (Paggio, 1998).

Lexical recall gives the percentage of valid words correctly accepted by the spelling checker, error recall indicates the percentage of invalid words correctly flagged and precision gives the percentage of correct flags (correctly found invalid words) over all flags by the spelling checker. With all metrics it holds that numbers are higher when performance is better.

There is a difference between a spelling checker and a spelling corrector. This difference is often recognised, but in practice hardly even made. A spelling checker searches text to find spelling errors. A spelling corrector not only finds spelling errors, but also suggests possible corrections for the errors.

In practice, most spelling checkers are actually spelling correctors.[3] From this point of view, we should evaluate the suggestion accuracy of the spelling correctors. However, the development of an improved suggestion module is currently still under construction and will not be treated in

this article. Some more information on the results on the suggestion capabilities of the spelling correctors can be found in (Van Zaanen and Van Huyssteen, 2003).

2.2.2 Evaluation Problems

When evaluation spelling checkers, it should be clear that the specific results obtained depended heavily on the used texts. Different texts will generate different results, since a number of words in the text are (or are not) part of the lexicon. This is not the only variable in the evaluation of spelling checkers. This section discusses some of the notions and problems of the evaluation of spelling checkers.

The evaluation approach taken in this article is as follows. A text, called the *test text* is given to the two spelling checkers. This text contains valid and invalid words. The number of found valid and invalid words of both spelling checkers is counted and these counts are then used to determine the performance effectiveness of the spelling checkers, by using the formulas in Table ???1.

The recall and precision results of the spelling checkers are only comparable when they are applied to the same text. This implies that comparing the results obtained in this article to results given in other publication (that are generated by applying spelling checkers on different texts) are not directly comparable. This also happens, only worse, when comparing spelling checkers of different languages.

Another problem is related to the composition of the test text. The test text should contain known valid and invalid words. If a large text is written for this, it may contain errors itself (because it has been written by hand or extracted from a corpus). Extreme care has to be taken to make sure that all spelling errors are known. A related problem is to decide what exactly is being evaluated. One could concentrate on specific features of a spelling checker (e.g. its capability to handle productive compounding, proper names, abbreviations, or other morphologically complex words). Examples of these features should then be contained in the test text. All the choices about the construction of the test texts should be taken into account when computing (and comparing) the recall and precision of the spelling checkers.

Currently, most (commercial) spelling checkers are unable to handle context-sensitive spelling correction. We consider the correction of multi-word units (such as fixed expressions or idioms) as context-sensitive, as well as the correction of words in context such as *wie* (‘who’) or *wat* (‘which’ – used as relative pronouns), or homonyms such as *vlei* (‘flatter’) and *vly* (‘lay down’). We have not evaluated this type of error, but in a more general evaluation, this could be taken into account.

In the current evaluation, we have applied the spelling checkers to a text containing 1337 different words, of which 1255 are valid and 82 invalid. For this evaluation task, the text has

been constructed based on a small Afrikaans corpus of e-mail messages, and contains simple words, morphologically complex words, compounds, proper names, abbreviations, and foreign words that are valid in Afrikaans. Invalid words from the corpus have also been included.

The evaluation in this article is not meant to show which spelling checker is better, but merely to investigate what improvements are wanted.

2.2.3 Results

This section will describe some results of the two evaluated spelling checkers, *PUKspell* and *Bspell*. The results of applying the spell checkers to the test text described above, can be found in Table 2. This table contains percentages as computed by the metrics of Table ???1 and, additionally, the plain counts are given between brackets. Note that further results can be found in Van Zaanen and Van Huyssteen (2003).

Measure	PUKspell	Bspell
Lexical Recall	95% (1196/1255)	98% (1225/1255)
Error Recall	84% (69/82)	83% (68/82)
Precision	54% (69/128)	68% (68/98)
Suggestion Adequacy	???	???

Table 2: Comparison of evaluation results

The actual results indicate two interesting differences between the two spelling checkers. The numerical results indicate that *PUKspell* performs overall worse than *Bspell* on this text. This is mainly because it rejects too many valid words, hence the lower lexical recall and precision. This can possibly be explained by the fact that the lexicon of *Bspell* is 31% larger than that of *PUKspell* (respectively approximately 180,000 and 235,000 words). The other difference can be found when looking at the rejected words. Both spelling checkers have difficulty handling compounds. Afrikaans is a semi-agglutinative language, so compound formation is productive.

The results indicate two means of improving the existing spelling checker:

- Increase the number of valid words the spelling checker will accept. This can be done by increasing the lexicon or adding morphological information.
- Extend the spelling checker with compound analysis. Related to this improvement is the handling of morphologically complex (non-compound) words.

2.3 Improving the Existing Spelling Checker

This section describes several improvements that are implemented in the existing spelling checker. The improvements are selected based on the results described in the previous section. First, an overview of the existing spelling checker is given, followed by a similar overview of the improved spelling checker. Next each improvement is discussed separately. Finally, some remaining problems are mentioned briefly.

2.3.1 Overview of the Existing Spelling Checker

The architecture of the currently available version of *PUKspell* can be found in Figure ???1. First, *Microsoft Word* delivers an arbitrary piece of text to be checked. This piece of text is then tokenized by breaking the text up in words on spaces. The lexicon lookup module then tries to find each word in its lexicon. This lexicon consists of roughly 185,000 words. If it is not found, the token is handed to the suggestion module, which searches for words in the lexicon that are relatively close to the incorrect word. The similar (correct) words are given to the user.

[INSERT FIGURE HERE]

Figure ???1: Architecture of the existing version of *PUKspell*

Since *PUKspell* is developed for *Microsoft Word*, the interface between the word processor and the spelling checker is fixed and outside our control. *Microsoft Word* fills an array with characters and then applies the spelling checker on this array. The text in the array does not have to be a single word, a sentence, or entire paragraph. It can contain one or more words, possibly including sentence boundaries, but this can be different each time the spelling checker is called. This is why the text needs to be tokenised first.

Since the spelling checker is mainly a lookup program that searches for words in a lexicon, the success of the spelling checker is directly related to the completeness of the lexicon. If not enough words are contained in the lexicon, too many valid words are flagged as invalid.

There are two opposing views on how large a spelling lexicon should be (Vosse, 1994, p.49). One view claims that the lexicon size should be kept small, because increasing the lexicon size results in more misspelled words to be accepted. This happens in particular when infrequent correct words that differ only in one letter with another valid word are added to the lexicon. The other view points out that the lexicon should be large, because this will reduce the number of valid words that are flagged as invalid. Based on the results, we concluded that the lexicon should be larger.

2.3.2 Overview of the Improved Spelling Checker

This section will give an overview of the architecture of the improved spelling checker. Figure ???2 shows the improved spelling checker in a graphical format. Each of the modules will be described in some detail next.

[INSERT FIGURE HERE]

Figure ???2: Architecture of the improved version of *PUKspell*

In the improved spelling checker, the lookup module consists of several steps. First, the word is looked up in the standard lexicon. If it is not found, it does some error detection tests. This consists of two steps, lookup in the invalid word lexicon and n-gram analysis. If the word is still found to be invalid, it is send to the suggestion module (shown as a dashed arrow in the figure).

The error detection lookup module (invalid word lexicon and n-gram analysis) can only decide that a word is invalid. If the word is not recognized in this module, it is handed to the morphological analysis module. This module strips affixes and checks if that leads to a valid word.

The idea behind stripping affixes is that morphologically complex words are not stored in the lexicon, but by reducing the morphological complexity of the word, a simple word remains that *is* part of the lexicon. The heart of this phase is a Porter stemmer (Porter, 1980; Kraaij, 1994) for Afrikaans. This stemmer has been developed at the PUCHE (???Van Huyssteen, 2003).

Note that usually a stemmer is used in the context of information extraction and retrieval. In the IE/IR context, words that are in the same semantic class should reduce to the same stem. This is not the case here, where valid words should simply remain valid (and invalid words must also remain invalid).

The main difficulty with using a stemmer is that it can easily over-generate. For example, Combrink (1990) identifies the *a-* as a possible prefix in Afrikaans, for example in words as *asosiaal* ('asocial') and *atonaal* ('atonal'). However, this prefix cannot be used in the spelling checker because the prefix cannot be removed in words like *Afrikaans* and *artistiek* ('artistic'), where the *a-* is clearly not a prefix.

If the word is still not found, it is given to the suggestion module. Currently, the suggestion module is a simple extension of that of the existing spelling checker. However, as mentioned earlier in this article, improving the suggestion module is future work.

2.3.3 Improvements

In this section, we will concentrate on the improvements that are implemented in the improved spelling checker. The section starts with the improvements of the lexicon. First, the enlarged lexicon is described, followed by the morphological analysis module. Finally, the error detection module is treated.

???@@@

2.3.3.1 Enlarged Lexicon

The evaluation of the two spelling checkers showed us that too many valid words are not recognized. In the improved version of the spelling checker, we tackle this problem in two ways. In this section we describe a direct extension and in the next section a more indirect extension of the lexicon.

The easiest way to get the spelling checker to accept more valid words is simply to increase the size of the lexicon. In the next release of the spelling checker, the lexicon will be 37% larger; containing some 250,000 words. The lexicon size is then on par with that of *Bspell*.

However, increasing the lexicon does not entirely solve the problem. Even though more valid words are accepted, there is a structural difficulty with the generative lexical power of agglutinative languages. The next section will discuss a partial solution of this problem.

2.3.3.2 Morphological Analysis

Afrikaans has many morphological rules that can be used to create new words (AWS, 2002). Of course, one can add all words that can possibly be generated to the lexicon; the spelling checker will then accept all these words, but this will increase the size of the lexicon exponentially. However, if this is done, the structural information that is inherently incorporated in morphologically complex words (and that can be described by simple rules) is discarded.

The approach taken in the improved spelling checker is to incorporate morphological information in the form of morphological rules. This will keep the size of the actual lexicon small, but by applying the rules, morphologically complex words (that are not directly in the lexicon) are still considered valid. In other words, adding morphological rules has two advantages. The size of the lexicon is reduced, while on the other hand, the number of accepted words grows.

Figure ???3 depicts the relationships between the size of the lexicon of the spelling checkers and the accepted words. The middle ellipse delimits the valid words accepted by the existing spelling checker. The smallest ellipse is the size of the original lexicon after removing all morphologically complex words. When morphological rules are added to the lexicon, all words in the largest ellipse are accepted.

Note that in the improved spelling checker, the lexicon containing morphologically simple words is expanded, this results in a lexicon that is larger than the lexicon of the existing spelling checker.

[INSERT FIGURE HERE]

Figure ???3: Morphological rules increase the number of accepted valid words

2.3.3.3 Error Detection

The improved spelling checker contains two new modules that handle error detection. These modules are added to speed up the lookup process, such that the analysis of more complex words later on in the process does not have to be extremely efficient (because not all invalid words reach that part of the process).

The first error detection modules is the *error lookup* module. It is a lookup module that searches a lexicon containing *invalid* words. Together with the invalid words, their best suggestions (one or more) are stored. When a word is found in this phase, the spelling checker can automatically give these suggestions to the user. An example word in this lexicon is: *inteligent* with *intelligent* as its suggestion.

A second way of filtering out obviously invalid words is done by using n-gram analysis. We have extracted n-grams (3, 4, and 5-grams of *characters*) from a corpus of plain text (of roughly 150,000 words). Tokens that contain n-grams that were not found in the corpus are considered invalid. This flags clearly invalid words such as *xyyz*, but it also finds words, such as *maatxkappy* ('society'), where an *x* is (incorrectly) substituted for an *s*, because they are next to each other on the keyboard.

2.3.4 Remaining Problems

The improved version of the spelling checker for Afrikaans has many improvements over the existing version. However, some problems still remain. This section will focus on these remaining problems. First, we will look at handling compounds and, secondly, something will be said about the suggestion module.

2.3.4.1 Analysing Compounds

In section 2.3.3.2???, we explained that adding morphological information to the spelling checker increases the number of valid words that can be handled. These morphological rules are implemented using a Porter stemmer. The stemmer can strip affixes, but is unable to handle compounds.

Finding the separate elements in compounds is a difficult problem. There are several reasons for this. For example:

- A compound can consist of several words, e.g. *strand-hand-doek-stof* ('beach towel cloth'), so trying to combine only two words from a lexicon is not enough.
- Words can have infixes, such as *-s-* in *bruilof-s-gas* ('wedding guest'), *-e-* in *student-e-nommer* ('student number'), or *-en-* in *wa-en-huis* ('garage'). But also *-ns-, -ens-, -tes-*, and *-er-* are possible infixes in a compound.
- Sometimes, doubling of consonants occurs, for example in *den-n-e-boom* ('pine tree'), where the *n* is doubled.
- A compound can also be constructed from morphologically complex words, for example *vergadering-s-prosedure* ('meeting procedure') or *verantwoordelikheid+sin* ('sense of responsibility').

At the moment, we use a longest prefix and suffix string matching to find elements in a compound (Van Huyssteen, 2003). However, this does not work well with morphologically complex words, since these are handled by the stemmer. In the end a complex interplay between finding the word boundary and using the stemmer might solve this problem, when taking added letters and the doubling of consonants into account.

A more interesting approach would be to use machine learning. Using a lexicon of structured compounds (delimiters between the elements of compounds), it is possible to train a classifier. New words are then handed to the classifier, which will insert the word breaks.

There has already been some work on compound splitting (Koehn, 2003), but most approaches make extensive use of structured training data. Unfortunately, this data is not (yet) available for Afrikaans, making these approaches less applicable.

Another problem, which is strictly related to handling compounds in a spelling checker, is that when a compound contains an error, it might not be so easy to find the compound boundaries. Previous work has concentrated on finding word boundaries within valid compounds. Spelling errors in a compound can result in incorrect word boundaries. Once incorrect word boundaries are found, it might prove difficult in the end to correct the error in the compound.

2.4 Summary

To decide which improvements over the existing spelling checker are wanted, it is compared against another commercial spelling checker for Afrikaans. This showed that precision and recall should be improved as well as the suggestion module.

By adding morphological information, we try to enhance the generative power of the lexicon and subsequently recognize more words. To reduce the number of accepted invalid words, n-gram analysis and an error lexicon is added. This will ultimately lead to an increase in precision and recall.

3. Teaching and learning

It was indicated in the previous section that a wealth of experience and knowledge are??? gained through this project. As little (or none) literature exist on Afrikaans computational linguistics (with specific reference to text technology), students and lecturers/researchers have to learn together, even by making mistakes and taking "wrong" sidetracks. This hands-on approach has proved to yield excellent results in the acquiring of knowledge in the field of computational linguistics.

This section describes how the graduate and post-graduate teaching programmes at the PUCHE benefit from this spelling checker project. First, it is indicated how work on the project is integrated in the graduate programme, with possibilities to incorporate future work in the programme. We then give an overview of the involvement of post-graduate students in the project, and how they gained knowledge by working on this project. In the last instance, it is illustrated how members of staff and students alike acquired knowledge (and experience) outside the boundaries of the curriculum, by getting trained in specialised skills.

Since 2002 a new graduate programme was introduced at the PUCHE, called *BA Language Technology*. The introduction of this programme was motivated by two factors, viz.:

(a) the need to develop teaching programmes that are relevant, vocationally directed, and future-oriented; and

(b) a gap in the South African educational market, with no graduate programmes on offer in computational linguistics.

After wide consultation with international and local role-players and experts, a programme was designed that combines language subjects, subjects from computer science, mathematics and statistics, as well as a core group of computational linguistic subjects (e.g. Introduction to Computational Linguistics, Methodology and Formalisms, Corpus Linguistics, Statistical Natural Language Processing, etc.). From the onset, the curricula of the latter group of subjects were envisaged to be problem-oriented and project-organised, based on the educational system developed at the Aalborg University, Denmark, since 1974 (Kjersdam and Enemark, 1994).

Problem-oriented education (or Problem Based Learning – PBL) can be defined as learning "based on working with unsolved, relevant and current problems from society/real life… By analyzing the problems in depth the students learn and use the disciplines and theories which are considered to be necessary to solve the problems posed, i.e. the problem defines the

subjects and not the reverse" (Kjersdam and Enemark, 1994: 16). By means of this applied research and teaching approach a dynamic triangular equilibrium between research, education and practice is maintained, serving researchers (and research outputs), students, and the industry alike. Also, by enabling students to gain "comprehensive knowledge of the development of theoretical and methodological tools" (Kjersdam and Enemark, 1994: 17), they will, after completion of their formal studies, be able to contribute to research and the development of original paradigms to solve new and complex problems in the future.

PBL is incorporated with project-organised education in two ways. On the one hand various project-based modules are included in the curriculum. For instance in the third year of study, the modules "Speech Technology: Applications" and "Text Technology: Applications" are introduced, where students will develop various small modules of both speech and text technological applications (e.g. a simple rule-based sentenciser or tokeniser). In the final year of study, the largest part of the year is spent on independent project work, which is conducted either at the university, or while doing an internship elsewhere. The idea is that these projects will be on a much larger scale than the third-year projects, possibly building on the work in the previous year (e.g. to develop a more sophisticated sentenciser or tokeniser, using more advanced NLP techniques).

On the other hand, course work in some of the other modules (e.g. in "Corpus Linguistics" or "Speech Technology: Introduction") is also organised around existing research projects. Students are mostly involved on a so-called "design-oriented" level, i.e. where they have to deal with "know-how problems which can be solved by theories and knowledge they have acquired in their lectures" (Kjersdam and Enemark, 1994: 7). After the project and the problems related to the project are explained to students, they get involved by collecting data, identifying possible/different solutions, formulating rules and algorithms, analysing data, evaluating different components, etc. In this way they therefore get know-how and experience on theoretical, methodological, and implementation level.

For instance, in the modules "Grammar: Introduction" and "Grammar: Intermediary", students were introduced to this spelling checker project. It was explained to students why the project was started, and what the problems were with the existing spelling checker (i.e. that it was not able to deal with the morphological productivity of Afrikaans). After introducing some basic and general concepts of morphology (e.g. inflection, derivation, compounding, etc.), students got involved in the project by collecting data (e.g. compounds) and analysing this data, using the theoretical framework that was introduced during the lectures (i.e. Cognitive Grammar and Construction Grammar). The outcome of this course work was several descriptions of different morphological constructions in Afrikaans (e.g. the plural construction, the past tense construction, etc.), which was implemented in the stemmer that was developed. Also, new data could be added to the lexicon of the spelling checker.

The advantages of this approach proved to be numerous: not only did the project benefit from the data collection and descriptions, but students got the feeling that they were involved in "important" and relevant work. They got the opportunity to apply the theoretical knowledge they acquired in the classes in a practical, hands-on environment, and in that way improving their understanding of the theories and concepts of the study field. Also, of course, our general understanding of Afrikaans morphological structure (especially within a computational context) was enriched by means of this approach.

On post-graduate level, the honours module "Language, text, and technology" in the Afrikaans honours programme, was, among other modules, introduced to provide students with a general BA or B.Sc. background, an entrance point into post-graduate studies in computational linguistics. From its very inception, this module was conceptualised to be problem-oriented and project-organised, offering lecturers an opportunity to align course material with their research projects.

In 2002, three students enrolled for the module, and the curriculum was organised around the spelling checker project. Together with these three students we first discussed the project, its aims, and its possible outcomes in immense detail (i.e. students were involved on the design-oriented level), thereby applying their theoretical knowledge of aspects such as text editing and Afrikaans grammar in a practical and useful way. After that, each of the students got involved in solving different problems of the projects, e.g. evaluating the spelling checkers, developing a stemmer and a word segmenter, extending and cleaning-up the lexicon, etc. Although each of them worked separately on different problems, they were forced to extend their experience by helping each other with their different tasks, thereby expanding their general knowledge and experience.

Once again, the advantages of this approach proved to be multitudinous. Students and lecturers/researchers with no background in computational linguistics got the opportunity to obtain knowledge and experience in the field, while learning to solve real-world problems. Additionally, members of staff were enabled to harmonise their research and teaching responsibilities, optimising the quality and quantity of their outputs. By involving students as co-workers we were able to get work on the spelling checker, as well as on the development of computational linguistic resources for Afrikaans, done in a shorter time. Moreover, all three students were motivated to continue with their studies in computational linguistics on MA level, where they are currently involved in the NRF-funded project on Afrikaans computational morphology.

Learning in this project was, of course, not restricted to the formal educational programmes, as extensive learning took place outside the curriculum. For example, the linguists in the team (students and researchers) had no (or very little) experience in computer programming. However, during the course of this project, they were all forced to learn to formalise their ideas

and knowledge, in order to communicate effectively with the other members of the team (i.e. the computational linguist and the programmers). They therefore had to learn more about formulating algorithms, drawing up flow charts, and, in general, thinking like computer programmers do.

More importantly, during the project, the linguists were forced to learn how to do actual programming in order to develop prototypes of the stemmer and word segmenter. This was done by learning to use regular expressions, implementing it in the Perl programming language. Perl is a programming language that was initially developed specifically to do text processing, and is therefore ideally suited for the computational linguist working with text. Furthermore, it is a relatively easy programming language to learn, and is therefore considered to be a good first language for inexperienced programmers to learn. During the course of this projects, the linguists in the team therefore acquired programming knowledge and experience that will be invaluable for future projects.

Related to the above, was the learning involved in working in a project team. The team consisted of several linguists, a computational linguist, and a computer scientist (programmer), with administrative, legal and financial support by people not working in either of the above mentioned fields. It was therefore of utmost importance to ensure that all parties involved understand exactly what was expected from him/her, by communicating ideas in an effective way. This project was thus an attractive way for linguists to get a good feeling of particular problems that arise when implementing linguistics in a computational environment, while learning to express their ideas in a structured, logical way. The computer scientist also had to learn to deal with data and structures that are atypical of the data that he was used to working with. In order to facilitate these communication problems, the computational linguist played a vital role in training the other groups to understand the idiosyncrasies of the respective fields.

To conclude: this spelling checker project offered us the opportunity to integrate research and educational activities, by taking a problem-oriented and project-organised approach to education. Students and researchers /lecturers worked together on the development of computational tools and resources for Afrikaans, while simultaneously learning about computational linguistics, natural language processing, and other related activities (i.e. programming, working in project teams, etc.). Not only did this approach proved to be an effective way of learning, but also to be efficient in financial terms and in terms of time.

4. Conclusion

Even though many computational linguists are not particularly interested in researching spelling checkers, implementing a spelling checker that performs well proves to be a difficult problem.

In this article, we give an overview of a spelling checking project for Afrikaans. This project has several aspects that are all discussed in detail. The project should be seen as a first project of a computational linguistics (sub-)department at the University of Potchefstroom. This university is the first to offer a complete computational linguistics course in South Africa.[4]

The computational linguistics students are considered part of the project from an early point. This means that they participate directly in the development of the spelling checker. Not only do they see what problems occur, but also how these can be solved.

At the end of the project, the existing spelling checker for Afrikaans will be improved in several ways. In this article, the existing spelling checker was compared to another commercially available spelling checker for Afrikaans and based on these results, improvements are proposed.

Notes

[1] We thank and acknowledge the following members of the research team for their inputs in this research: Roald Eiselen, Christo Els, Petri Jooste, Christo Muller, Sulene Pilon, Martin Puttkammer, Werner Ravyse, and Daan Wissing. We would also like to express our gratitude towards Attie de Lange, Ulrike Janke, Boeta Pretorius, and Elsa van Tonder for technical, administrative, and legal support. The PUCHE also sponsors this project generously – our thanks to Frikkie van Niekerk for his support.

[2] Due to South African law, we are not allowed to use the name of the commercial spelling checker directly.

[3] In this article, we will not make a real distinction between the terms spelling checker and spelling corrector.

[4] Other universities in South Africa have single courses on specific computational linguistics topics, but no complete courses are given.

References

Kjersdam, F. & Enemark, S. 1994. *The Aalborg Experiment: Project Innovation in University Education*. Aalborg: Aalborg University Press.