Detection of Duplicate Defect Reports Using Natural Language Processing

Conference Paper *in* Proceedings - International Conference on Software Engineering · June 2007

DOI: 10.1109/ICSE.2007.32 · Source: IEEE Xplore

3 authors, including:

Per Runeson
Lund University
233 PUBLICATIONS **15,131** CITATIONS

SEE PROFILE

Detection of Duplicate Defect Reports Using Natural Language Processing

Per Runeson, Magnus Alexandersson and Oskar Nyholm
Software Engineering Research Group
Lund University,
Box 118, SE-221 00 Lund, Sweden
per.runeson@telecom.lth.se

Abstract

Defect reports are generated from various testing and development activities in software engineering. Sometimes two reports are submitted that describe the same problem, leading to duplicate reports. These reports are mostly written in structured natural language, and as such, it is hard to compare two reports for similarity with formal methods. In order to identify duplicates, we investigate using Natural Language Processing (NLP) techniques to support the identification. A prototype tool is developed and evaluated in a case study analyzing defect reports at Sony Ericsson Mobile Communications. The evaluation shows that about 2/3 of the duplicates can possibly be found using the NLP techniques. Different variants of the techniques provide only minor result differences, indicating a robust technology. User testing shows that the overall attitude towards the technique is positive and that it has a growth potential.

1. Introduction

When a complex software product like a mobile phone is developed, it is natural and common that software defects slip into the product, leading to functional failures, i.e. the phone does not have the expected behavior. These failures are found in testing or other development activities and reported in a defect management system [5][18]. If the development process is highly parallel, or a product line architecture is used, where components are used in different products, the same defect may easily be reported multiple times, resulting in duplicate reports in the defect management system. These duplicates cost effort in identification and handling, hence support to speed up the duplicate detection process is appreciated.

The defect reports are written in natural language, and the duplicate identification requires suitable information retrieval methods. In this study, we investigate the use of Natural Language Processing (NLP) [17] techniques to help automate this process. NLP is previously used in requirements engineering [12][3][19], program comprehension [2] and in defect report management [15], although with a different angle.

Basically, we take the words in the defect report in plain English, make some processing of the text and then use the statistics on the occurrences of the words to identify similar defect reports. We implemented a prototype tool and evaluated its effects on the internal defect reporting system of Sony Ericsson Mobile Communications which contained thousands of reports. Further, we interviewed some users of the prototype tool to get a qualitative view of the effects. The prototype tool identified about 40% of the marked duplicate defect reports, which can be seen as low figure. However, since only one type of duplicate reports are possibly found by the technique, we estimate that the technique finds 2/3 of the possible duplicates. Also, in terms of working hours, reducing the effort to identify duplicate reports with 40% is still a substantial saving for a major software development company, which handles thousands of defect reports every year.

The paper is outlined as follows. Section 2 introduces the theory on defect reporting and on natural language processing. Section 3 presents the tailoring made of the NLP techniques to fit the duplicate detection purpose. In Section 4, we specify the case study conducted for evaluation of the technique, and Section 5 presents the case study results. Finally Section 6 concludes the paper and outlines further work.

2. Theory

2.1 Defect Reporting

Whenever a problem is found during development, testing or operation, it has to be notified to the one responsible for identification of the underlying fault and its correction. To help with this, Defect Management Systems (DMS) or "bug tracking" systems are used [5][18]. With this tool, the tester, or whoever discovers a problem, can submit a Defect Report (DR).

A general workflow for how defect reports are handled follows:

- A failure or an issue is found and a report is created in the defect management system.
- The report is analyzed by an analyst to isolate the cause.
- The report is assigned to a developer for correction by a change control board or similarly.
- The developer finds the defect and corrects it.
- The fix is tested by the developer or by a test department.
- The report is closed.

Defect report management is similar to task management in general, e.g. in a service organization. The report is a kind of "relay" in relay race, intended to solve a task. Different actors contribute to solving the task, and the information management system is the central node which dispatches subtasks to the actors.

A defect report must contain information about the origin, the defect as such and software under test. The scheme used in our context is similar to one, proposed by Pol et al [16], see items Table 1. Our context uses the DMS also for change requests, hence the types are defect report and change request.

When we search for duplicate DRs, part of the problem lies in defining what counts as a duplicate DR. There are two types of duplicates; 1) those that describe the same failure and 2) those that describe two different failures with the same underlying fault. See Figure 1. These two kinds are inherently different in that the former type, which describes the same failure, generally uses the same vocabulary, while the latter type, which describes two failures stemming from the same fault, may use different vocabulary. The technique we use in this study is based on vocabularies, and hence, we address only defects of type 1.

As an example, there might be two defect reports stating that when you push the *back button* on the phone, nothing happens. These would count as duplicate reports describing the same failure. The reports will probably use similar vocabulary.

Table 1. Information items in a defect report

Item	Explanation
Header	Concise description of the problem[1]
Project name	Which project the DR concerns
Number	An unique identification number for this DR
Tester	The name of the tester
Date	The date of submission
Urgency	How important it is that this is fixed
Type	Whether it is an defect report or a change request[2]
Test object	What software the test was performed on
Version	What version of the software was used
Test specification	Reference to the test case used
Description	A description as to what went wrong
Appendices	Attachments such as test logs etc.
Remarks	Comments

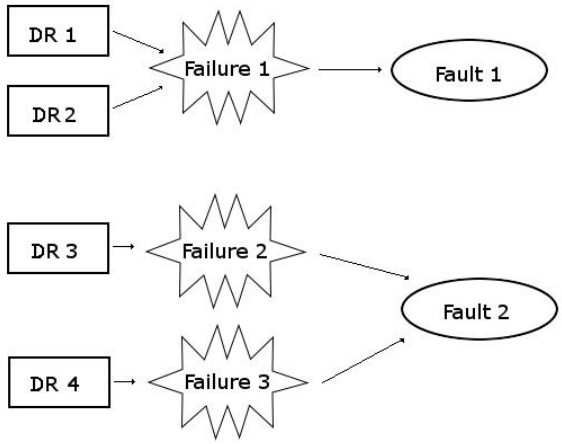

Figure 1. The two different types of DRs. DR 1 and DR 2 describe the same failure which stems from Fault 1. DR 3 and DR 4 describe two different failures which both in turn stem from Fault 2.

Another example is one report stating that there is something wrong with the standby time of the phone, and another saying that the phone keeps loosing reception. These two reports might have the same source,

[1] This field is not in the scheme proposed by Pol et al.
[2] Pol et al's scheme depicts type to software, specifications, documentation or technical infrastructure.

e.g. that the code that controls the antenna is faulty. If the problem with the antenna is solved, the standby and reception failures would both be solved. They are therefore also counted as duplicate reports, but do not use the same vocabulary since the failures, on the surface, are very different.

The first submitted DR on a specific fault is denoted the *master report*, and the subsequent reports on the same fault are called *duplicate reports*.

Although much effort is spent on defect handling in software engineering, very little research is done on the tools for defect management. Recommendations on the introduction and use of DMS are recently published [5][18]. However, to the best of our knowledge, the only research published on the DR duplicates is the work by Podgurski et al [15], who cluster related defects, using information from automatically reported software failures.

2.2 Natural Language Processing

The goal of information retrieval research is to develop algorithms and models for retrieving information from document repositories. This information is mainly textual, expressed in Natural Language (NL) or structured NL. The classical information retrieval problem is called the ad-hoc retrieval problem. Here the user defines the information he is looking for with a query, and the system returns a list of documents. An exact match system demands that the documents in every detail satisfy some structured query expression. Manning and Schütze [8] claim that for large and heterogeneous documents collections, this might lead to that the list of results might either be empty, or very large. Recent work has therefore concentrated on ranking documents according to relevance to the query, i.e. finding the list of most similar documents.

The processing stages in NLP are described below based on Manning and Schütze [8]. They comprise:

- tokenization,
- stemming,
- stop words removal,
- vector space representation, and
- similarity calculation.

2.2.1 Tokenization. Tokenization means turning a stream of characters into a stream of tokens. This is done by removing capitals, punctuation, brackets etc. Basically each token is a word, although the definition of a word is not straightforward. A word might be defined as a string of alphanumeric characters surrounded by white space. But there are several alternatives on how to treat hyphens and apostrophes and other punctuation marks.

Most punctuation marks, like commas and semicolons, are easy to remove, as they clearly are not part of any word. Periods however, are used to mark both end of sentence and abbreviations such as *etc.* Apostrophes sometimes cause confusion, for example, it is unclear whether *boy's* is the possessive case or an abbreviation for *boy is* or *boy has*.

Hyphens may either be split into several words or kept together. Some words like *e-mail* are best kept as one word, while others like *so-called* are more open for debate. A different kind of hyphens is those inserted to help indicate the correct grouping of words, for example *a text-based medium*. It is common, like in this case, to hyphenate compound pre-modifiers.

Different policies can be chosen regarding how to split into words, and the choice depends on the type of data to tokenize.

2.2.2 Stemming. Stemming aims at identifying the ground form of each word. Words may be written in different grammatical forms, but still carry the same information. During this phase affixes and other lexical components are removed from each token, and only the stem remains. For example, *worked* and *working* are both transformed into *work*. Verbs are also transformed to their ground form, e.g. *was* and *being* become *be*.

2.2.3 Stop Words Removal. There are many common words, like *the, that* and *when*, that do not carry any specific information and hence are not likely to be of any help in the similarity analysis. These words occur in all texts with approximately the same frequency, and do not relate to the content of text. While they are semantically important, if not removed they could disturb the similarity calculation. Most of them are prepositions, conjunctions or pronouns.

Therefore a list containing those “stop words” can be applied to the text. All words matching the stop words list are removed. Exactly what words to include on the list depends on the type of data [8].

If the texts are based on a template, it might be beneficial to remove the words that make up the template to reduce these words’ impact on the similarity measure.

As an alternative to stop words removal, the terms may be weighted based on the inverse frequencies of the word in the total corpus [9]. The more frequent a word is, the less information it carries, and hence the less it should be weighted in the similarity calculation.

2.2.4 The Vector Space Model. The next step is to represent the words in a multi-dimensional vector space model. Each dimension of the space corresponds to a word. The position along each axis in this space depends on the frequency of the word occurring in the text. The similarity between two texts is then measured in terms of distances in this vector space.

The dimensions must not necessarily be linear, i.e. a simple word count. A word that occurs three times is probably more important to the content of the text than another word that only occurs once, but not three times as important. Therefore the term frequency needs to be dampened. A common approach is to use a weighted scale, as defined in equation (1):

$$weight = 1 + log(frequency) \quad (1)$$

This analysis may be extended to pairs or triples of words. This increases the computational complexity, but is shown successful in other applications [9].

2.2.5 Similarity Measures. The calculation of the similarity between two texts is done on the vector space model. The three most common measures are *Cosine*, *Dice* and *Jaccard* [8]. According to Salton, "The choice of a particular vector-similarity measure for a certain application is not prescribed by any theoretical considerations, and is left to the user." [17].

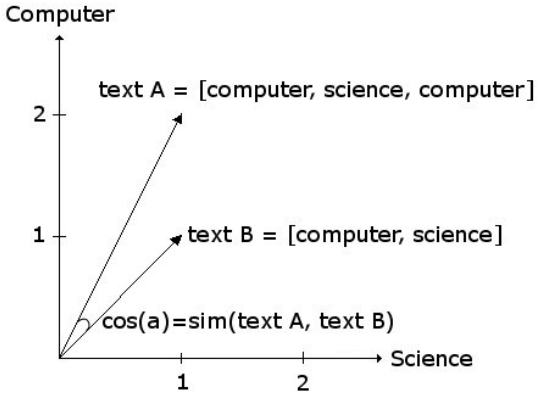

Figure 2. Similarity between the two texts *computer science computer* and *computer science* using the Cosine measure and the vector space model.

All three measures are normalized, in order to take into account the length of the vectors. A graphical representation of a similarity measure calculation is shown in Figure 2.

2.3 NLP in Software Engineering

Software engineering is mostly based on natural language information in different kinds of specifications and other documents. Hence, the field is a candidate for using NLP techniques. Published studies using NLP in software engineering are mainly in the requirements engineering and program comprehension subfields.

Natt och Dag et al [12][13] link two different kinds of requirements together, using NLP. Market requirements, which express customer wishes on future products, are linked to the company's internal requirements in a compliance checking procedure. They have developed a prototype tool – ReqSimile[3] – to support the linking. The approach is empirically evaluated in industrial case studies [12][13]. Natt och Dag et al also used the tool to identify duplicate requirements, which application was evaluated in an experiment [14].

Hayes, Dekhtyar and Sundaram [3][4] use NLP to support tracking of requirements to designs. They have developed a requirements management tool, called RETRO, which supports the tracking. They report on evaluation of different NLP variants in their paper.

Yadla, Hayes and Dekhtyar [19] have linked requirements to defect reports in order to support the defect correction process. They investigate relevance feedback in the learning of NLP algorithms. This work is implemented as a part of the RETRO tool. The authors mention in this paper identification of duplicate defect reports as a piece of future work.

Lormans and van Deursen [6] used latent semantic indexing (LSI) to automatically reconstruct traceability links between requirements and design, and requirements and test cases respectively. They evaluated two alternative approaches empirically in three industrial case studies.

Antionol et al [3] traced C++ source code onto manual pages and Java code to functional requirements. They use both a probabilistic information retrieval model and vector space models.

Canfora and Cerulo [2] searched for source files through change request descriptions in open source code projects. They find change requests being good indicators for impact analysis on where to change the code.

Maarek et al [9] supported searching in a software reuse library. They created an indexing scheme, using NLP techniques, to help programmers find reuse candidates in the code library.

[3] http://reqsimile.sourceforge.net

3. NLP in Duplicate Detection

In this section, we present how we tailor the NLP techniques to identify duplicate defect reports. We have developed a tool, based on the core functionality of the ReqSimile tool. This section presents the choices with respect to NLP techniques, and Section 4 presents the case study evaluation.

3.1 Tokenization, Stemming and Stop Words

The tokenization, stemming and stop words removal are made based on Minnen et al's work [10]. Almost all periods are removed, except periods in floating point numbers and in references to standards, like *ISO-8859-1* [12]. All other punctuation marks are discarded. This means that hyphenated words without digits are split into several tokens. All capital letters are transformed into lower-case.

In the stemming, all affixes are removed, which is quite straightforward.

The stop words list consists of words commonly used with stop lists, like articles and prepositions, plus words we found that occur frequently in the defect management system. Examples of these are *attach* and *log*, which are used in many DRs, but say nothing about the defect that is reported. See an example in Table 2.

Table 2. Results of tokenization, stemming and stop words removal

Original sentence	Tokenization	Stemming	Stop words removal
Dump in file system when recording audio (see attached log).	dump in file system when recording audio see attached log	dump file system when record audio see attach log	dump file system record audio

3.2 Synonyms and Spellchecking

When dealing with large amounts of raw text submitted by many different persons, it is natural that differences exist in which words are used to signify a given meaning. Based on this observation we decided to implement a simple thesaurus.

In order to construct the thesaurus, we took the 1.000 most frequently used words in the DMS, and studied what words were used interchangeably with the words in the list. We also asked employees at the company to give us a list of words, which were used interchangeably.

It is worth noting that not only pure synonyms were considered, but also words that are used in the same context and mean almost the same. Examples of this are *crash* and *dump*; a *crash* means that the phone stopped responding while a *dump* means that it stopped responding but also a snapshot of the memory could be recovered. While these two might not actually be synonyms, they are used interchangeably when writing DRs. Abbreviations such as *bt* for *Bluetooth* were also added to the thesaurus.

The thesaurus is applied after the stop words removal. It checks all tokens if they are present in the thesaurus and substitutes them with their synonyms.

We also made a simple spellchecker which works in the same way as the thesaurus, i.e. it recognizes misspelled words and substitutes them with the correctly spelled ones instead.

The basis of the spellchecker comes from a list of usually misspelled words[4] and we used much of the same method as with the thesaurus. Spellchecking is performed at the same time as substitution of synonyms. The result after synonym replacement and spellchecking is shown in Table 3.

Table 3. Results of synonym replacement and spellchecking

Stop words removal	Synonym replacement
dump file system record audio	dump file system record sound

3.3 Similarity Measures

To measure the similarity value between two DRs, we use the vector-space model along with the cosine measure. Other measures were tested (Jaccard, Dice), but did not improve the result and were hence discarded. All term weights are dampened by equation (1).

3.4 Information Elements

A DR consists of many fields, together carrying a lot of information, see Table 1. Two of the fields are written in natural language; the description field which thoroughly describes the problem, and the header which summarizes it. A decision has to be made of what fields shall represent the DR as a text. We started up with header and description combined, but also evaluated including project name and assigning a

[4] http://www.wsu.edu/\~brians/errors/misspelled.html

higher weight to the header in the similarity calculation.

3.5 Time Frame

One problem with the duplicate detection approach is that when searching for a duplicate to a specific master DR and the set of possible candidates, i.e. the rest of the DRs, is very large; the duplicate must be very similar to the master to come up in the result list since there are so many DRs that are compared. One way of countering this is to narrow the search to fewer candidates. This also improves the calculation performance.

We investigated narrowing the search, using only the DRs launched within a certain time frame. Figure 3 shows a diagram over the time difference between when a duplicate and its respective master was submitted.[5] As can be seen in the figure, 53% of the duplicates were submitted within 20 days after its corresponding master. 90% are in the range between 20 days forward and 60 days back.

This led to us implementing what we call a *time frame*, which can be specified by the user. If there are two years worth of DRs in the database, using a time frame of fifty days back and ten days forward, the number of candidates are reduced to 5%, leading to reduced calculation costs. The evaluation of the use of the time frame is presented below.

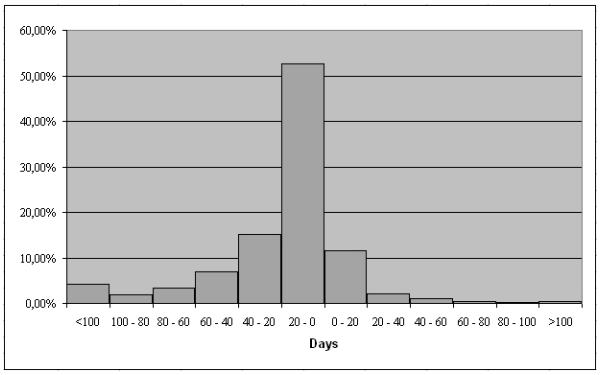

Figure 3. Time difference between when a duplicate and its master were submitted

3.6 Top List Size

The NLP algorithm ranks the document by similarity. However, in order to be a useful tool, only a subset can be presented to the user. The top list size is defined as the number of possible candidates that are returned from a duplicate search. As the candidates are to fit into a window and be reviewed manually we use top list sizes of 5, 10 and 15 DRs in the evaluation.

[5] A duplicate may be reported as being submitted *before* its master, since the assignment to master/duplicate is somewhat arbitrary. In the analysis, we always consider the oldest DR being the master.

4. Case Study

4.1 Environment

Sony Ericsson Mobile Communications develops mobile multimedia devices for GSM and UMTS standards. It is a large company with a complex software development process. Some 20 unique products are launched every year. Sony Ericsson uses a Software Product Line approach when developing mobile phones. The notion is to have one single platform for a family of phones. They all share this single platform, and upon it, modules are added as needed. These modules could be seen as building blocks and the platform as the foundation. Depending on which building blocks are laid on the foundation, the resulting product will exhibit different characteristics.

While this is an efficient way to develop new phone models, it is also a good source of duplicate DRs. Whenever code is reused in new products and different releases, great care has to be taken to fix defects in the right version of the code; otherwise the failure will still be present in the latest version or, in the version that actually exhibited the failure.

Handling of these defect reports are managed through a DMS. The database comprises thousands of defect reports from previous and current development projects. Some 10% of the DRs are marked as duplicates. Comparing a new report to already existing reports, in hope of finding a duplicate, is a tedious and error prone process. The current search engine in the DMS is a basic string matcher to which you can pass additional arguments such as time interval and DR id interval.

4.2 Methodology

In order to evaluate the NLP technique for DR duplicate identification, we used two approaches. Firstly, batch runs were conducted of the duplicate detection procedures against the database and secondly, user tests by testers and analysts were conducted, followed by interviews.

We evaluated different parameters in the NLP technique, changing one factor at a time [11]. This simple experimental design is assumed to be sufficient at this stage, since we have not seen any indications on major interactions between the factors. The evaluated variants are:

- Similarity measures: Which of them works best for duplicate DR detection?
- Stop words list: What impact has the number of words in the list, and is it useful at all?
- Time frame: How old DRs should be included in the search?
- Synonyms and spellchecking: Does a basic approach increase the recall rate?
- Fields: What fields should be passed on to the preprocessor? Is it a good idea to include the project on which the report is submitted? Since the header describes the problem quite concise, should it be weighted more than the regular description?

All the variants were tested on three different top list sizes.

4.3 Evaluation Measures

The two most important evaluation measures in information retrieval systems are *recall* and *precision* [8]. Generally speaking, recall is the percentage of relevant items selected out of all the relevant items in the repository, while precision is the percentage of relevant items out of those items selected by the query.

These metrics do not really fit into how duplicate detection works. Consider a search for a duplicate where the size of the top list is set to 10. What you are looking for is *one* DR, namely the corresponding master DR. A successful search would thus yield a precision of 10% since one item out of ten was relevant. Recall would then be a binary value, 100% or 0%; either it is found or not.

Natt och Dag et al. instead uses *recall rate* to measure how efficient their requirements similarity tool works [12]. Recall rate is here defined as the percentage of duplicates for which the master is found for a given top list size. This answers the question of *What percentage of the duplicates is found that today are marked as duplicates in the DMS?*

This evaluation measure is conservative, since we have used only the DRs marked as duplicates as basis for the batch runs. The recall rate might very well actually be higher for two reasons; 1) all duplicates are probably not identified and marked as such in the database, and 2) the NLP technique is relevant for identifying only duplicates with common failure behaviour, as discussed in Section 2.1.

5. Evaluation Results

5.1 Batch Runs

The batch runs were conducted as follows. We selected the DRs in the DMS that were marked as duplicates and each corresponding master report. For each duplicate DR, a similarity search was performed and the position of the duplicate DR in the top list was observed. From this data, the recall rates were calculated for different top list sizes. The same set of DR data is used for all the tests. The size of the data set is counted in thousands, although exact figures cannot be reported for confidentiality reasons.

To evaluate the different NLP factors, we set up a number of test cases. The baseline setup of parameters is to use the cosine measure, search 50 days back, use the small stop words list, use synonym replacement, use spellchecking and include product name.

The results are presented as recall rates for different sized lists. Note that the left column is the same for all tests.

Table 4. Recall rate for different similarity measures

Top list size	Cosine	Dice	Jaccard
5	0.3127	0.3024	0.3024
10	0.3822	0.3749	0.3749
15	0.4240	0.4153	0.4153

As seen in Table 4, Dice and Jaccard give the same result. This is always the case; their ranking is the same, even if they provide different similarity values. The cosine coefficient shows a slightly better result, and continues to be our main choice.

Table 5. Recall rate for different time frame

Top list size	50-0	60-0	100-0	500-0
5	0.3127	0.3114	0.2990	0.2485
10	0.3822	0.3831	0.3719	0.3084
15	0.4240	0.4219	0.4149	0.3434

There is a very small difference between using an interval of 50 days and 60 days, see Table 5. Expanding the interval to 100 days does decrease the recall rate some, but the difference is not significant. Further expansion to 500 days can not be recommended, due to the heavy decrease in recall rate.

Table 6. Recall rate for different stop words lists. The small stop words list contains 60 words, and the big list contains 439 words.

Top list size	Small	Big	None
5	0.3127	0.3055	0.2811
10	0.3822	0.3719	0.3451
15	0.4240	0.4101	0.3834

The small stop words list, with 60 words, gives the best result, see . While the difference in result is not that big between the two stop words lists, it can be seen that without any stop words, the result becomes significantly worse.

Table 7. Recall rate using synonyms

Top list size	With synonyms and spellchecking	Without synonyms and spellchecking
5	0.3127	0.3068
10	0.3822	0.3813
15	0.4240	0.4233

By using synonym replacement and spellchecking, the result is improved very little, see Table 7. It is impossible to judge whether this indicates that synonym replacement and spellchecking is not necessary or that the basic approach we have used is not efficient enough.

Table 8. Recall rate using project field

Top list size	Using project field	Not using project field
5	0.3127	0.3016
10	0.3822	0.3731
15	0.4240	0.4097

As expected, including the project name increased the recall rate some, see Table 8. It turns out that weighting the header, so that it is included twice in the search, also increases the recall rate somewhat, see Table 9.

Table 9. Recall rate using double header

Top list size	Not using double header	Using double header
5	0.3127	0.3172
10	0.3822	0.3867
15	0.4240	0.4238

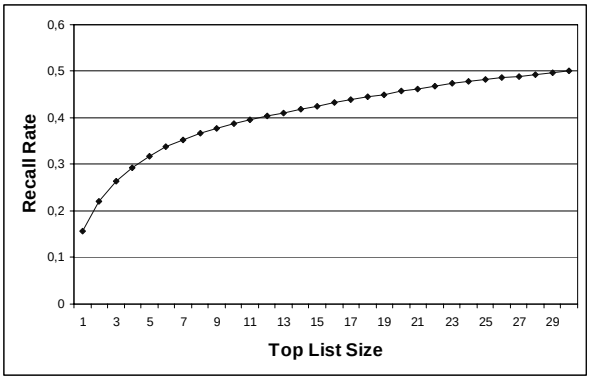

Figure 4. Graph showing the recall rates for different sizes of the top list.

As can be seen in the result tables above, the best result we get is finding approximately 39% for a top list size of 10 and 42% for a top list size of 15. As outlined in Figure 1, not all duplicates actually describe the same defect, and are therefore not possible to detect with the NLP approach. Another problem is DRs that are irrelevant for the search, but still get a high similarity value. Thus they push relevant DRs from the result list. There are several reasons why they might get a high similarity value:

- Unnecessary tagging. Some users write information in the header and description areas that could be found in other fields of the DR. For example, defect severity.
- Users write all their DRs in a similar way and probably copy texts sometimes. Therefore, when searching for duplicates to a DR of theirs, other DRs written by the same user will fill up the list.
- Several different problems can be described using the same vocabulary.

further indicates that not all duplicate DRs can be found using the NLP approach. The asymptotic recall rate for large top list sizes is not 1, but rather between 0.5 and 0.6, indicating that an estimated maximum of 60% of the duplicates can at best be found by the technique. As can be seen in the graph, the curve is steep in the beginning but flattens as the top list size grows larger. In effect, this means that those duplicate DRs that use similar words will be found. Those that are not similar will, regardless of top list size, never be found. Hence our results indicate a recall rate of about 2/3 of the possibly detectable duplicates.

5.2 Interviews

To evaluate the usefulness and other qualitative aspects of the duplicated detection, we interviewed one tester that used the prototype tool and a team of analyst.

5.2.1 Interview with tester

The tester found the prototype tool being was helpful in his day-to-day work in finding duplicates. He thought that "any approach to making it easier to find duplicates was good".

The tester had mostly used the tool to search duplicate DRs based on typing in a few keywords, which is not the best suited task for the NLP technique. He thought that it would be hard to get testers to use the tool in its intended way, i.e. enter a full DR and then search for duplicates. When you have decided to write a DR, you have already done all the research necessary to submit a DR, and then you will do so.

The biggest future improvement would be to integrate the tool into the DMS. Other improvements that could be made is more ways of limiting the search such as; being able to search only from one organizational group's DRs and to search only from DRs concerning one specific project.

5.2.2 Interview with analysts

The analysts had actually not used the prototype tool as such. Instead, one of them had taken out the core functionality from the tool and incorporated it into a tool that he used when analyzing DRs. In his tool, he had added a button that whenever he analyzed a DR, he could push this button and get a top list containing the most likely duplicates. He had then given this modified tool to his colleague who also had started using it.

The analyst found that the technique worked very well for them. One thing in particular that they thought worked very well was to search for duplicates to somewhat "unclear" DRs, i.e. DRs that are not easy to understand what they actually mean. One of the analysts said that it saved time due to when she was to search for duplicates the ordinary way, she had to sit down and really understand the DR, then she had to think up words that she thought described the DR and search with those words. Using their modified tool she could just push a button. If she found a duplicate she was done, otherwise she of course had to do it the ordinary way.

As one of them said, even if I do not find a duplicate with this approach, it takes so little time to push the button and skim through the result list so the extra effort does not matter.

5.3 Costs and Gains

Finding 40% of the marked duplicate DRs may be seen as a very low figure. Still, if the technique helps eliminating 4% of all DRs (assuming 10% duplicates as in our case) this is an enormous saving for a major software developer. For confidentiality reasons, we cannot report the total number of DRs and hence the exact savings.

However, Sony Ericsson estimated that 30 minutes were spent on average to analyze a DR that is submitted. Using the NLP technique, a duplicate would be found in 30 seconds. For this case, 20 hours of analysis time would be saved per 1000 DRs, since on average 40 duplicates can be found by using the NLP-based tool.

6. Conclusions and Further Work

The aim of this work is to evaluate the feasibility of using Natural Language Processing techniques to help automate detection of duplicate defect reports.

We evaluated the identification capabilities on a large defect management system and concluded that about 40% of the marked duplicates could be found. This figure is rather stable for different variants of similarity measures, stop words lists, spellchecking and weighting scheme. The major difference is the length of the top list – the chance is of course higher that the duplicate is in a top list of 15 than one of 5. Relating to the estimated maximum recall rate, about 2/3 of the duplicates are found.

The users found the technique helpful in their work, although they had to change their way of searching compared to traditional key word search. One finding from the interviews is that the duplicate detection rather is a task for defect report analysts, rather than for testers. Even though only 40% of the duplicates are found, this approach can provide a substantial saving for major development organizations.

Further work includes integration of the support into the DMS and the defect handling processes of Sony Ericsson. Regarding the technology as such, more alternatives remain to be evaluated, such as using a corpus based weighting factor and pairs or triples of words [9]. Comparing the approach to general search engine techniques or plagiarism checkers would give relevant reference points for the study. Replications on other data sets would also be interesting to study.

7. Acknowledgements

The authors are grateful to Mr. Stefan Qelthin of Sony Ericsson for providing access and guidance to the DMS and relevant personnel. Thanks to Dr. Johan Natt och Dag for willingly answering questions on the ReqSimile tool. Thanks to the anonymous reviewers for constructive comments, especially on further work and to Dr. Björn Regnell for additional review. The work is partly funded by the Swedish Research Council under grant 622-2004-552 for a senior researcher position in software engineering.

8. References

[1] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, 28(10):970-983, 2002.

[2] Canfora, G. and Cerulo, L., "Impact Analysis by Mining Software and Change Request Repositories", *Proceedings 11th International Software Metrics Symposium*, pp 29-29. 2005.

[3] Hayes, J. H., Dekhtyar, A., Sundaram, S. K., "Improving After-the-fact Tracing and Mapping: Supporting Software Quality Predictions", *IEEE Software*, 22(6):30-37, 2005.

[4] Hayes, J. H., Dekhtyar, A., Sundaram, S. K., "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods", *IEEE Transactions on Software Engineering* 32(1):4-19, 2006.

[5] Johnson, J. N. and Dubois, P. F., "Issue Tracking", *Computing in Science and Engineering*, pp. 71-77, November/December, 2003.

[6] Lormans, M. and van Deursen, A., "Can LSI help Reconstructing Requirements Traceability in Design and Test?", *Proceedings of the Conference on Software Maintenance and Reengineering*, pp. 47-56, 2006.

[7] Macias, B. and Pulman, S. G., "Natural Language Processing for Requirements Specification", In *Safety Critical Systems*, pp. 57-89, Chapman and Hall, 1993.

[8] Manning, C. D. and Schütze, H., *Foundations of Statistical Natural Language Processing*. Cambridge, USA: MIT Press, 1999.

[9] Maarek, Y. S., Berry, D. M. and Kaiser, G. E., "An Information Retrieval Approach for Automatically Constructing Software Libraries", *IEEE Transactions on Software Engineering*, 17(8):800-812, 1991.

[10] Minnen, G., Carroll, J. and Pearce, D. "Robust, Applied Morphological Generation", *Proceedings of the First International Natural Language Generation Conference*, Mitzpe Ramon, Israel, pp. 201-208, 2000.

[11] Montgomery, D., C., *Design and Analysis of Experiments*, 5th Edition, John Wiley and Sons, 2000.

[12] Natt och Dag, J., Gervasi, V., Brinkkemper, S. and Regnell, B. "Speeding up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering." *Proceedings of the 12th International Requirements Engineering Conference*, pp. 283-294, 2004.

[13] Natt och Dag, J., Gervasi, V., Brinkkemper, S. and Regnell, B., "A Linguistic-Engineering Approach to Large-Scale Requirements Management" *IEEE Software*, 22(1):32-39, 2005.

[14] Natt och Dag, J., Thelin, T. and Regnell, B., "An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development", *Empirical Software Engineering* 11(2):303-329, 2006.

[15] Podgurski, A., Leon, D., Francis, P., Masri, W. and Minch M., "Automated Support for Classifying Software Failure Reports", *Proceedings of the 25th International Conference on Software Engineering*, pp. 465-475, 2003.

[16] Pol, M., Teunissen, R. and van Veenendaal, E., *Software Testing A Guide to the TMap Approach*. Harlow, England: Pearson Education, 2002.

[17] Salton, G., *Automatic text processing: the transformation, analysis, and retrieval of information by computer*, Reading, MA: Addison Wesley, 1989.

[18] Serrano, M. and Ciordia, I., "Bugzilla, ITracker, and Other Bug Trackers", *IEEE Software*, 22(2):11-13, 2005.

[19] Yadla S., Hayes, J. H. and Dekhtyar, A., "Tracing requirements to defect reports: an application of information retrieval techniques", *A NASA Journal, Information Systems Software Engineering* 1:116-124, 2005.