Eindhoven University of Technology
Department of Mathematics and Computer Science

Master's thesis

Related Defect Report Detection using an Information Retrieval Approach

October 4, 2013

Author: ing. Y.M. Schroot
y.m.schroot@student.tue.nl

Supervisor: dr. A. Serebrenik
a.serebrenik@tue.nl

Tutor: ir. V.A. Kalloe, PDEng

Assessment committee: dr. A. Serebrenik
ir. V.A. Kalloe, PDEng
dr. B.F. van Dongen
prof.dr. M.G.J. van den Brand

Abstract

Defects, also known as bugs, are a major issue in the world of software development. Developing complex bug-free software is a major challenge. Fixing defects is very time-consuming, therefore different approaches were introduced for improving the defect fixing process. A commonly used method is duplicate defect detection. Using this method duplicate defect reports are automatically recognized and marked as duplicate in the repository. As a consequence, the number of unresolved bug reports is reduced.

However, by applying duplicate defect report detection the number of defect reports decreases, but the time to resolve a particular defect is not reduced. According to some experts at Océ, analyzing the defect report takes most of the time. Therefore, we suggest reducing the time of solving the defect by presenting a list of related defect reports. Especially for less experienced defect solvers this approach could save a lot of time.

We consider related defect reports as are already considered defect reports, which could support the defect fixer in solving a newly reported defect. A defect regarding a text alignment problem could, for instance, support the bug fixer in solving a defect concerning an image alignment problem. Because the notion of ‘related defect’ is vague, the opinion of the developers was taken into account.

To detect related defect reports, we applied natural language processing techniques. In contrast to duplicate detection, defects occurred on other products than the newly reported defect could also be related. Consequently, we only evaluated the short summary (synopsis) and description of the defect report.

We present two algorithms which compute, for a given defect, a top ten list containing related defects using the synopsis and/or description. Both algorithms are implemented in a tool called Related Defect Finder (RDF) and evaluated in a case study at Océ Technologies. To achieve a maximum performance in terms of speed, we indexed all terms of the synopses and descriptions.

RDF was able to compute a top ten result list (based on the synopsis) in less than half a second in a repository containing over more than 40,000 defect reports. Because the best performing algorithm was only based on the synopsis, we also introduced an optimized algorithm. This optimized algorithm computes the results based on a combination of the synopsis and description in less than six seconds on average.

Our experimental results show that using the synopsis-based algorithm at least one related defect in a top ten list can be detected in 76.9% of all cases. Furthermore, in 41.7% of all cases a related bug reported showed up at the first position of the result list. On average 26.4% of the defects in the top ten list is related. The optimized algorithm performed a little bit better. On average 28.4% of the defects in the top ten list was related, which is an improvement of 2%.

During an evaluation period of one month the RDF tool was used by 28 unique users. Of these 28 users, only five users used the tool more than three times. This small number of active users can be explained by the fact that most of 28 users were curious people like project managers for which the tool was not useful. The two most active users used the tool for different purposes (related and duplicate defect detecting). According to these two users the tool is a good addition in the process of resolving defects. For the less frequent users the

tool was in some cases not useful for their tasks. Other less frequent users seemed to be experts who (in most cases) do not need a list of related defects to solve a new defect.

Acknowledgments

First of all, I would like to thank my tutor, Amar Kalloe and supervisor, Alexander Serebrenik, for their guidance, feedback and support during my master project. Furthermore, I would like to thank Boudewijn van Dongen and Mark van den Brand for taking place in my graduation committee.

I also would like to thank Huug Schenk, Pieter Verduin, Oana Dragomir and Anneke Jansen for assigning the related defects which I needed to continue my project.

Finally, I would like to thank my parents who encouraged and supported me over the years. Without their love it would not be possible to complete this study.

ing. Y.M. Schroot

Contents

1 Introduction

Defects are a common issue in the world of software. In commercial software defects could even lead to financial losses [Tim12]. Therefore, bug fixing is a very important part of the developing process. According to an expert at Océ, tracing the bug takes often more time than solving the bug. Tracing the bug is the process of locating the cause of the problem. In a small simplistic software application developed by one software engineer, often that particular engineer could trace the problem relatively fast. However, if a bug shows up in a complex software application in which more than 300 developers are involved, tracing the bug could take a lot of time.

If all 300 developers are experts of a specific part in the system, the bug could probably be traced relatively fast assuming that a developer recognizes that the problem occurs in his part of the code. However, in this case all 300 developers have to evaluate all bugs which is not efficient at all.

Such an distribution of experts is of course not realistic, indeed it is quite risky, especially for commercial organizations. If one of the developers quits the company, it can have large consequences. Another developer has to delve into the code and needs probably a lot of time to understand it.

To prevent a company from such scenario's and to encourage personal development, switching between different projects and positions is a common phenomenon at Océ. The advantage of this approach is that most developers at Océ gain knowledge about different subsystems and components. For instance, if someone has to ask for help, he/she is no longer dependent on just one person. As a result, less team members are needed to implement e.g. a new feature.

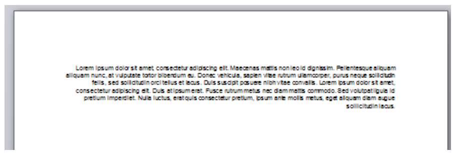
Using the approach at Océ, developers gain knowledge about a lot of different components within the system, but do not know all details within a particular component. As a result, tracing bugs could take some more time.

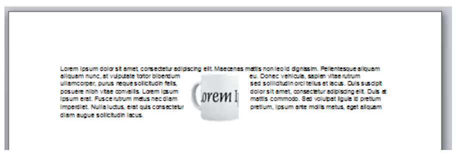
This would make it particularly beneficial to use the knowledge stored in the bug tracker. The bug tracker contains all bugs reported, and if solved, the associated solution. Newly reported bugs could be related to a bug occurred in the past. For instance, if a text alignment problem (Figure 1.1a) was reported previously and in the newly reported bug an image alignment problem was encountered (Figure 1.1b), then the solutions could be similar. According to experts at Océ, related defects could help the bug fixer in solving the problem.

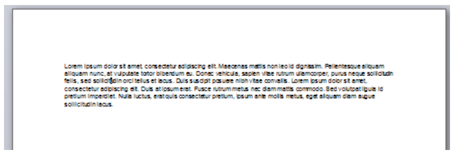
In this thesis we present an information retrieval approach to automate the process of detecting related bug reports. We develop a tool that presents a top 10 of defects possibly related to the bug provided by the user.

The remainder of this thesis is organized as follows. Chapter 2 gives an overview of the preliminaries. In Chapter 3 we discuss related work regarding duplicate defect detection. A pilot study was conducted to determine if related defects could be detected by comparing system logs. This pilot study is discussed in Chapter 4.

Chapter 5 describes the design which includes the approach to detect related defects. The natural language processing techniques relevant for this master project are described in Chapter 6. For each of the techniques it is indicated how the technique is applied in our project. Related work regarding the natural language processing techniques are also discussed in this chapter. In Chapter 7 the DRA (Defect Relation Allocator) tool is described. This tool is developed to evaluate our algorithms. First the approach and setup are discussed


(a) *Text alignment not correct.*


(b) *Image alignment not correct.*


(c) *Text alignment fixed*

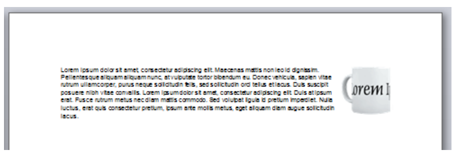

(d) *Image alignment fixed*

Figure 1.1: *Example of two different defects (Figures 1.1a, 1.1b) which are both related to alignment issues. In Figures 1.1c, 1.1d their corresponding fixes are shown.*

followed by the algorithms developed and evaluation of the performance of the DRA tool. At the end of this chapter the results including conclusions are presented.

In Chapter 8 the results of Chapter 7 are analyzed and five possible optimizations are introduced. Chapter 9 describes the RDF tool which is developed to detect related defects. In this chapter also the setup, requirements and performance of the tool are discussed. In the last section the results are presented. Finally, in Chapter 10 the conclusions of the this master thesis are presented. In this chapter, we also discuss directions for future work.

2 Preliminaries

2.1 Company

Océ is a Dutch company that develops, manufactures and sells printing and copying hardware and related software. Océ offers among others office printers, copying systems, production printers and wide format printers. Océ has its headquarters in Venlo (The Netherlands). In 2010 Océ was acquired by Canon.

2.2 Software configuration management

At Océ R&D IBM Rational Synergy is used as task-based software configuration management(SCM) tool in which artifacts related to software development are stored. This includes source code, images and documents.

IBM Rational Change is a web-based change management tool used to manage all feature requests and defects. Using Rational Change feature requests and defects can be created, updated and viewed.

2.3 Defect Report

When a defect is detected, a defect report has to be created. Figure 2.1 shows an example of a defect report. Due to confidentiality reasons some information is hidden or replaced by dummy data. We only discuss most relevant fields.

Each defect report contains a unique identifier, which is automatically generated by the system. The *synopsis* contains a short description of the defect. A comprehensive description of the defect can be given in the *description* field. In about $\frac{1}{3}$ of all defect reports a template is used for the *description* field. An example of an applied template can be seen in Figure 2.1. In this example the template is not filled in completely. During the years different kind of templates were introduced.

The next field to discuss is *status*. All statuses are described in Section 2.4. Two different requests are stored into the repository: Enhancements and Defects. We only focus on defects, therefore we are only interested in reports of request type 'defect'. *Severity* and *Occurrence* determine the *Priority*.

Keyword is an optional field and is used in about 21% of all defects. In general, this field is used to summarize the defect in a few words.

Work in DB describes at which Océ affiliate the defect was submitted (e.g. Venlo, Creteil etc.). *Found Product Name* states the name of the product on which the problem occurs, *Product Sub-System* indicates the subsystem in which the defect is encountered.

We do not discuss *Tree Release* because it is not relevant for this project. The same holds for *Release Note.* If a defect is detected by a customer instead of a tester, the external reference can be added in the corresponding field. When the defect is assigned to a developer, the developers name is entered in the *Resolver* text field.

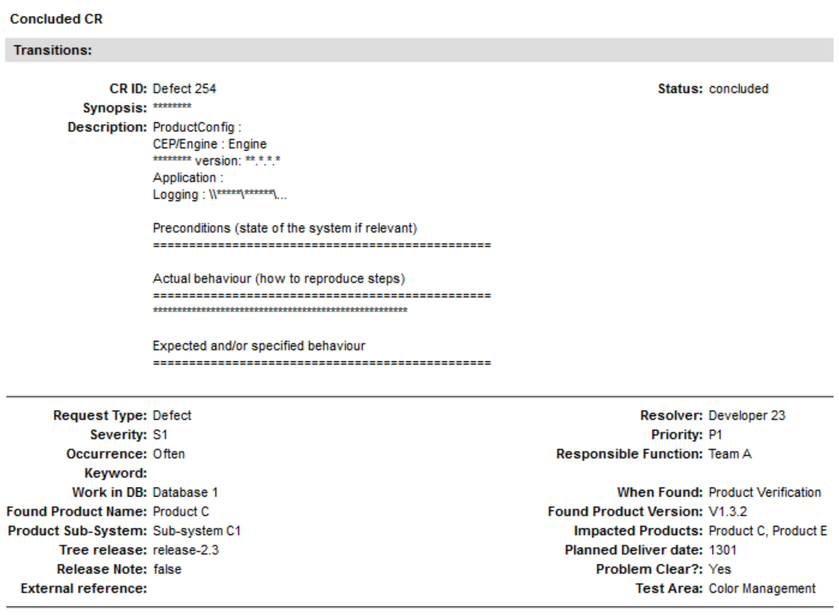
Responsible function tells which team is responsible for the defect. Only 23% of all defect reports are provided with a responsible team.

When Found tells us during which development phase the defect was detected. The version number of the product on which the defect occurred is saved in the *Found Product Version* field.

The field *impacted products* shows in which products the defect occurs. No strict rules are given to use this text field. As a consequence, sometimes full product names are used and sometimes only abbreviations are used.

Planned Deliver date tells in which month and year the defect is planned to be solved. '*Problem Clear?*' indicates if the resolver understands the problem. A simple *Yes* or *No* can be stored.

Test Area was introduced in January 2013. Therefore, only a small part of the defect reports (1.6%), contains this field.



Concluded CR

Transitions:

CR ID:	Defect 254	**Status:**	concluded
Synopsis:	********		

Description:

```
ProductConfig :
CEP/Engine : Engine
******** version: **.*.*.*
Application :
Logging : \\*****\******\...

Preconditions (state of the system if relevant)
================================================

Actual behaviour (how to reproduce steps)
================================================
***************************************************

Expected and/or specified behaviour
================================================
```

Request Type:	Defect	**Resolver:**	Developer 23
Severity:	S1	**Priority:**	P1
Occurrence:	Often	**Responsible Function:**	Team A
Keyword:			
Work in DB:	Database 1	**When Found:**	Product Verification
Found Product Name:	Product C	**Found Product Version:**	V1.3.2
Product Sub-System:	Sub-system C1	**Impacted Products:**	Product C, Product E
Tree release:	release-2.3	**Planned Deliver date:**	1301
Release Note:	false	**Problem Clear?:**	Yes
External reference:		**Test Area:**	Color Management

Figure 2.1: *Example of a defect report. Due to confidentiality reasons some information is hidden or replaced by dummy data*

2.4 Defect Life cycle

In this section we discuss all possible defect report statuses. A schema which shows the life cycle of a defect is shown in Figure 2.2. When a defect is detected by a tester, he/she submits a defect report into Rational Change. The status of the defect is initially set to *Entered.* After the defect is entered, the defect is analyzed by a team member of the given subsystem. If analysis takes a while, the status is changed by the reviewer to *In Review.*

If the defect is actually not a defect, the problem is rejected and the status is changed to *Probreject.* It could also be the case that the problem is indeed a defect, but the current behavior is accepted (e.g. if it is a scenario which occurs very rarely). In this case, the status is changed to *Accepted.* Sometimes a submitted defect was already submitted by another tester. The defect is in this case already present in the system and the defect is marked as *Duplicate.* If the analysis is ready—the impact is clear, the requirements are clear, the architecture is clear and the interface changes are clear—the status is changed to assigned.

When the defect is assigned to a developer, sometimes the problem is not clear to the developer or the developer could for example propose to reject the problem. In such cases the status changes from *Assigned* back to *In Review*.

If the developer indeed understands the problem, has resolved, documented and tested it, the defect is marked as *Resolved*. After a defect is set to resolved it has to be verified. If verifications fails, the defect report is reviewed again and its status changes to *In Review*. If the solution of the problem is verified successful, the defect report is marked as *concluded*.

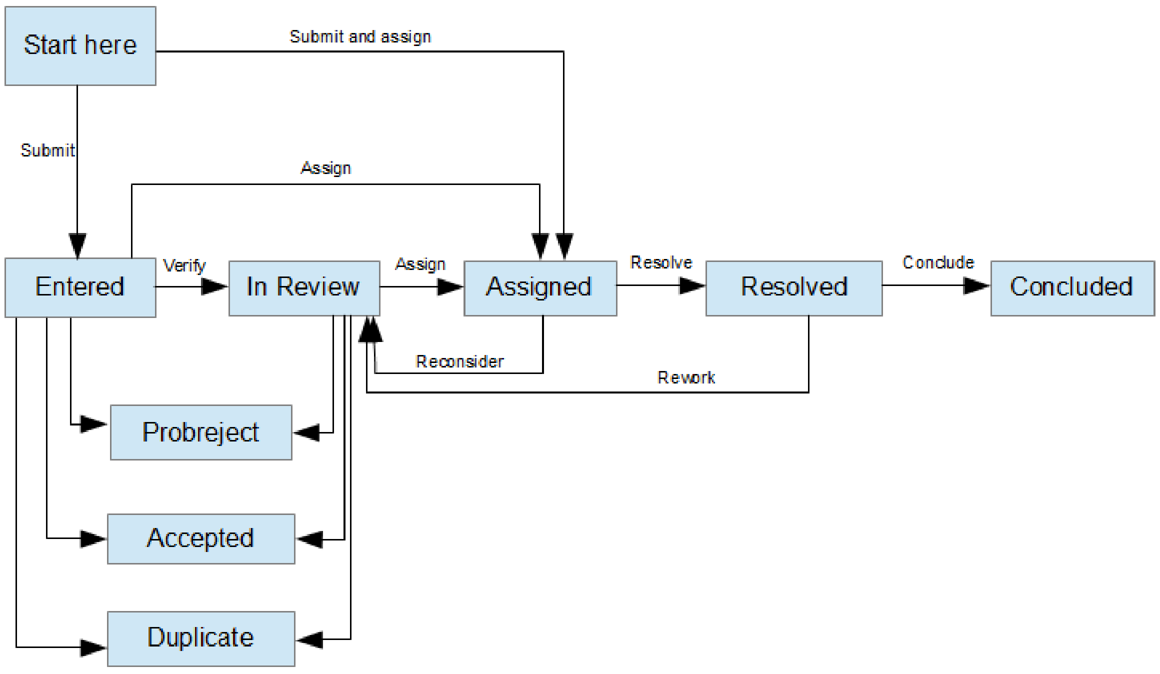


Figure 2.2: *Defect life cycle [Tec13]*

3 Related work

No earlier research was done on detection of "related defects". Related defects are defects which support a bug fixer by resolving a new defect. An approach to search for related defect reports is comparing the synopses (summary) and/or descriptions of two defect reports.

A topic which takes a similar approach is detection of duplicate defects. There are however significant differences between duplicate and related defect detection. In duplicate defect detection specific defect information such as product name, version and subsystem could be very useful. In related defect detection such fields are too specific and thus cannot be expected to be useful. Furthermore, in related defect detection the goal is to find as much as possible related defects, whereas in duplicate defect detection the goal is to find the master duplicate[1].

Work related to the applied techniques of pre-processing natural language documents, such as stemming and synonym replacement is discussed in Chapter 6.

Duplicate defect detection

In detection of duplicate defects a lot of research has been done [RAN07, JW08, WZX+08, BCA12, NNN+12, LM13]. Runeson, Alexandersson and Nyholm applied natural language processing (nlp) techniques to identify duplicate defect reports [RAN07]. Similarly to the defects at Océ, the defects considered contained a short synopsis and a more detailed description. The authors suggest to treat the synopsis as twice as important as the description. They develop a prototype tool and evaluate it in a case study at Sony Ericsson Mobile Communications. Evaluation shows that about 2/3 of all duplicates can be found using this approach.

Sun et al. applied a discriminative model for information retrieval including 54 features to detect duplicate bug reports [SLW+10]. The authors validate their approach on three different open source systems. Their approach improved a relative recall of 17-43% over other state-of-the-art techniques [JW08, RAN07, WZX+08].

Jalbert and Weimer propose a system that automatically classifies duplicate bug reports as they arrive [JW08]. Their system uses surface features, textual semantics and graph clustering to predict if the report is a duplicate. The authors evaluated their approach on the Mozilla project and were able to filter out 8% of the duplicate bug reports.

Wang et al. considered besides natural language processing also execution information to determine if a defect is a duplicate [WZX+08]. The proposed approach achieves a recall range of 67%-93% in the Firefox bug repository compared to 43%-72% using natural language information alone. Because execution information of Firefox was not available the authors had to generate all execution information manually. Therefore, in practice such an approach is often not useful.

Suraka and Jalote propose a character N-gram-based model to detect duplicate bug reports [SJ10]. In contrast to other approaches this approach is not word-based but based on characters. The advantage of this approach

[1]In a set of duplicate reports, the oldest report is the master duplicate.

is that it is language independent and robust towards noisy data. Using this approach the authors achieve a recall rate of 61.94% with a top 50 result list.

Sun et al. also encountered non-textual fields like product, component, version etc. to detect duplicate defect reports [SLKJ11]. Furthermore, they extended BM25F–a similarity measure, especially for duplicate report retrieval. This approach shows a relative improvement of 10-27% over their previous model [SLW+10].

In most approaches a written report is needed for determining if a defect is a duplicate. Lerch and Mezini presented an approach that only uses stack traces and their structure for detecting bug report duplicates [LM13]. This approach performed as good as natural language processing techniques.

Most duplicate defects quality performances are expressed in terms of recall[2], which is possible because the duplicate relationships are stored in the bug repository.

In contrast to 'duplicate' defects, 'related' relationships are usually not stored in the bug repository. As a consequence, recall values cannot be computed. Therefore, we have to express the quality performance using another measure.

[2]The fraction of relevant instances that are retrieved

4 Pilot study

4.1 Introduction

Detecting related defects can be done using different kind of techniques. The two most obvious approaches are comparing defects using natural language processing techniques applied on defect reports or applied on execution data (Chapter 3). A less popular approach is to use stack traces to determine if a defect is a duplicate. This could be done before creating the defect report[LM13] which saves a lot of time.

In this pilot study we investigated if execution data could be used to detect related defects. Logs are part of execution data and contain a lot of relevant data such as errors, system statuses, memory usage, executed functions etc. To determine if detecting related defects by comparing logs could yield meaningful results, we started interviewing three developers.

In those interviews we started by observing the process of fixing a defect. During the interview we focused on the different kind of logs (system log, trace log and protocol log) and their usefulness.

From the interview we could conclude that finding a duplicate defect was very difficult. Sometimes the interviewee recognizes the problem and remembers that a duplicate defect report exists including the corresponding solution. The problem is however that often duplicate defect reports could not be found by the developers. Therefore, developers also do not spend time on finding possibly related defects which could help them by solving the new defect; rather they start tracing the defect.

The interviews confirmed that manually finding related defects is a difficult and time consuming process. The logs seemed to be very suitable for detecting the related defects automatically. Wang et al. [WZX$^+$08] applied a quite similar approach. The authors compared execution information to detect duplicate defects. However, Wang et al. had to generate almost all execution logs manually, because the logs were not present in the repository. At Océ, all logs are stored on a log server. Therefore, using these logs for finding related defect reports could be very useful in practice.

4.2 Methodology

Figure 4.1 shows the general idea for finding related defects using log mining. First, the log is obtained from *defect report A* by extracting the associated log from the log server. Then, by applying a similarity algorithm to the log, a related log is identified. Finally, from the related log, the associated defect report B is obtained.

In order to compare logs, we first applied a regular expression to each of the logs such that we could extract timestamps, version numbers process names etc (Table 4.1). Then, all the extracted field are stored in a database. After collecting and storing the data we started analyzing the logs. During this phase we investigated which data was relevant to use in the similarity calculation and which was not.

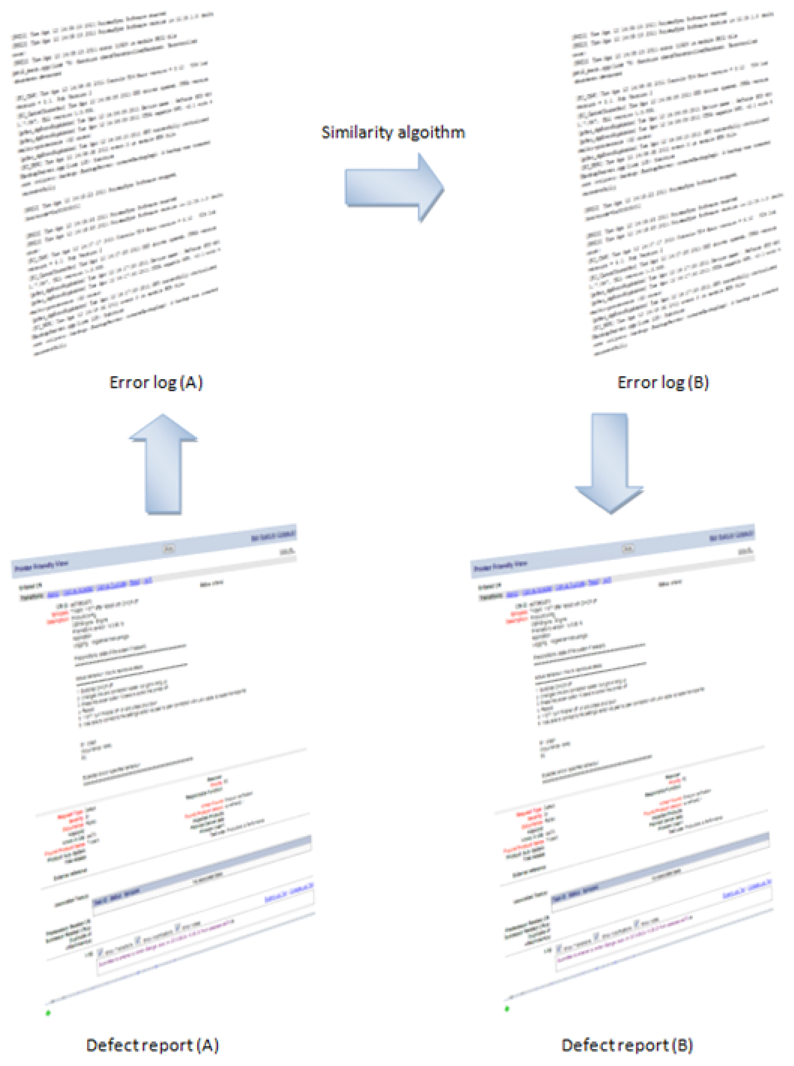


Figure 4.1: *Idea to find related defect reports using system log mining.*

4.2.1 Data collection

All log files are stored on a log server and each log is compressed into a password protected zip file. This zip file contains many different files such as templates, trace logs, protocol logs and the system log.

Initially, we focused on the system log. This log can be seen as a high level summary of all other logs. If an error occurred in a subsystem, then the error shows up in the subsystem's log. However, it will also show up in the system log. Most of the events only appear in the relevant subsystem's log and not in the system log.

Over more than 1,000 system log files are stored on the log server. Extracting all those zip files manually would take a lot of time. Therefore, we have written a little tool to extract and save the system log from the zip file automatically. After extracting all system logs we had to parse them, such that it could be stored in a database which makes similarity calculations in a later stadium more efficient.

Listing 4.1 shows an example of a system log. The extracted fields are shown in Table 4.1. Not all errors are registered in system logs. Therefore, it is not possible to find all related defects using the system log only.

Listing 4.1– Simplified example of an system log (id = *ehv*040#100.). Confidential information is replace by random strings.

```
[ABC1] Wed Apr 14 13:02:40 2010 Software started
[ABC1] Wed Apr 14 13:02:40 2010 Software version is 12.15.2.1 (multi core)
[XX_BBB] Wed Apr 14 13:02:46 2010 error 11504 in module BMM file abc_abc.cpp(line 102)
    function aaa::bbbbb::ccc::dddDD::attach: No bitmap exists with id=16
```

Table 4.1: *Extracted fields from defect id ehv040#100*

Defect id:	ehv040#100
Software version:	12.15.2.1
Process	XX_BBB
Message	No bitmap exists with id=16
Log line	102
Timestamp	2010-04-14 13:02:46
Filename	abc_abc.cpp
Module	bbbbb
Function	attach
Error id	11504

4.3 Results

In total 39, 628 defects were present in the Synergy database. Only for 5, 159 defects a system log could be matched, which is only 13% of all defects. However, if we only consider defect reports submitted after 1 January 2012, in 42% of the defects a system log was available.

In more than 20% of the system logs the error message was: 'Uncontrolled shutdown detected'. In most cases this message is a result of a tester not shutting down the controller properly. Therefore, these error messages are not suitable for finding system logs.

A solution to this problem would be to also take into account the trace (Listing 4.2) or protocol (Listing 4.3) logs. In total for each system log 45 different protocol logs and 49 different trace logs are stored. These level logs contain a lot more detailed information.

Listing 4.2– Fragment of a trace log. Confidential information is replace by random strings.

```
[..]
146500745382 04/19/11 10:53:47.828 t@03144 AAA XYZ-IPC.cpp(426){
    Java_com_oce_abc_Connection_connect}: Connect result 0, connection_handle 105E7560
147123386463 04/19/11 10:53:48.046 t@03144 BBB dummy.c(56){Java_com_oce_dummy_ABC_TraceMsg}:
    INFO [AWT-EventQueue-0] oce.a1.Loader - Attempting to load module mod.Module...
147413286103 04/19/11 10:53:48.156 t@03144 BBB dummy.c(56){Java_com_oce_dummy_ABC_TraceMsg}:
    INFO [AWT-EventQueue-0] oce.a1.Loader - Attempting to load module com.oce.mod3.
    controlmodule.defModule...
148289794816 04/19/11 10:53:48.468 t@03144 BBB dummy.c(56){Java_com_oce_dummy_ABC_TraceMsg}:
    INFO [AWT-EventQueue-0] oce.a1.Loader - Attempting to load module com.oce.mod2.abcModule
    ...
148919608161 04/19/11 10:53:48.687 t@03144 BBB dummy.c(56){Java_com_oce_dummy_ABC_TraceMsg}:
    INFO [AWT-EventQueue-0] oce.b1.LogBook - construct logbookmodel
[..]
```

Listing 4.3– Fragment of a protocol log. Confidential information is replace by random strings.

```
[..]
2012/09/14 07:30:14.465 [0001] [50140256] --> call  system_config {0} ::function()
2012/09/14 07:30:14.465 [0001] [50140256] <-- reply system_config {0} ::function(version =
    string "<version-number>") returns system_config {0}
2012/09/14 07:30:14.465 [0002] [50140256] --> call  system_config {0} ::function_register(
    name = string "<value>")
2012/09/14 07:30:14.465 [0002] [50140256] <-- reply system_config {0} ::register_client()
    returns session {1}
```

```
2012/09/14 07:30:14.465 [0003] [50140256] —> call  system_config {0} ::remove_ref()
2012/09/14 07:30:14.465 [0003] [50140256] <— reply system_config {0} ::remove_ref()
2012/09/14 07:30:14.777 [0004] [50139952] —> call  system_config {0} ::connect()
2012/09/14 07:30:14.777 [0004] [50139952] <— reply system_config {0} ::connect(version =
    string "<version-number>") returns system_config {0}
[..]
```

4.4 Conclusions

Only 13% of all defects was linked to a system log. This would mean that we can find a related defect for at most 1.3 defects out of 10. However, since January 7,627 defects are submitted between 01-01-2012 and 07-05-2013 of which 3,206 (42%) are linked to a system log.

Based on these numbers, 4.2 defects out of 10 are linked to a system log. On average 18.6 defect reports are submitted each day. This implies that at most 7.8 defects are linked to a system log each day. However, in more than 20% of the system logs between 01-01-2012 and 07-05-2013 the associated message was 'Uncontrolled shutdown detected' which makes these system logs unusable.

The lack of details in the system log also results in same error messages with totally different causes. Therefore, the probability to find a related defect will decrease even further.

Taking into account more detailed logs (e.g. trace log or protocol log) would be an option, but understanding these logs would take a lot of time. To understand these detailed logs a lot of domain and architecture knowledge is needed. It also gives no guarantees that it lead into better results. Therefore, we decided not to take the more detailed logs into account.

For defects in which a less popular error number is mentioned, finding related defects using the error messages result in very promising results. Unfortunately, these promising results only hold for a very small set of defects.

We decided, because of all these reasons, not to continue error log mining.

5 Design

As concluded in the previous chapter, log mining could only be beneficial for a small set of defects. Therefore, we decided to start a complete different approach which is often used in the domain of duplicate defect detection: natural language processing [RAN07, SLW+10, JW08].

5.1 Approach

We apply natural language processing techniques to find related defect reports. Each defect report contains a synopsis and a description. An example of a defect report is given in Figure 2.1. In our approach we search for related defects, therefore we do not take into account fields such as 'product name' or 'sub-system'. Indeed, two defects could occur on different products or subsystems, but the cause could be the same.

Normally, a training set is used to create and refine the developed algorithm and using a test set the quality performance of the developed algorithm is measured. However, in this particular case there is no complete data set available in which the relationship 'related' is stored.

Our intention was to evaluate all defects till 07-05-2013. In total $39,628$ defects were submitted during this period. To construct an oracle, we have to evaluate all possible pairs of defects performing in total $785,169,378$ comparisons (Equation 5.1). Manually evaluating these comparisons therefore is infeasible.

Calculation of the total number of equations.

$$\frac{\#defects^2 - \#defects}{2} = \frac{39,628^2 - 39,628}{2} = 785,169,378 \tag{5.1}$$

Consequently, creating a complete training and test set is not a realistic option. As an alternative we took a sample by randomly selecting 64 defects. By comparing all of the defect with each other we could create a smaller training set, which leads to $(64^2 - 64)/2 = 2,016$ comparisons. Unfortunately, after taking a quick look at the set with an expert it seemed that only a few defects were related to each other. As a consequence, the diversity of this set was too small, meaning that we could not use the sample to train and test our algorithms.

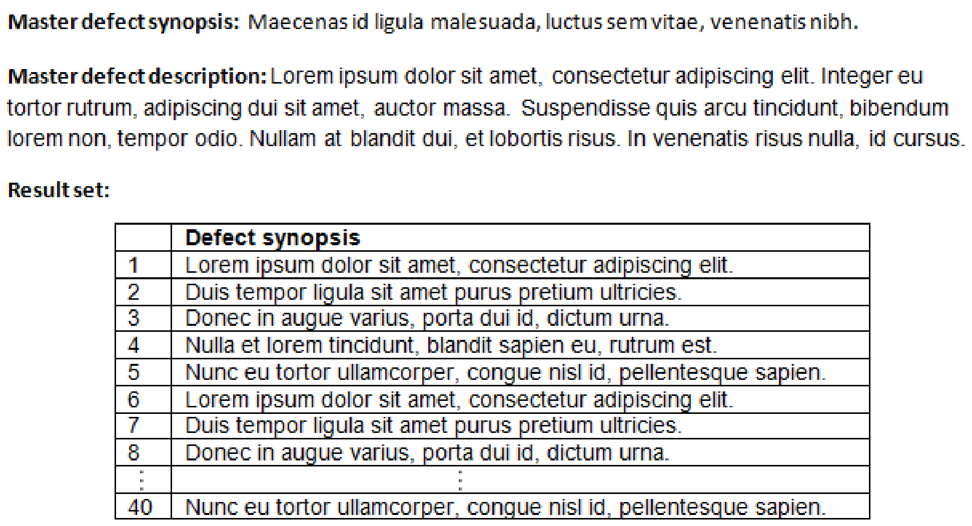
From this we concluded that the approach of creating a training and test set was not an option. We decided to develop three different algorithms and test their quality performance by implementing them in a tool and letting five experts evaluate the results. A random selection of defects is added to the results to prove that the algorithms perform better than a random selection.

Each algorithm returns 10 defects most closely related to a given defect. The random selection randomly selects 10 defects from the entire data set. Thus, in total the tool shows 40 results for each given defect:

- Top ten results of algorithm 1

- Top ten results of algorithm 2
- Top ten results of algorithm 3
- Ten randomly selected results

We asked five experts to determine for each of the results if it is related to the given defect. We refer to this given defect as the 'master defect'. Each of the experts is asked to evaluate the results of 10 different master defects. In total $10 * 40 = 400$ comparisons are evaluated by each of the experts. The results can be marked as 'related' or 'not related'. To prevent an expert from being influenced by the algorithm, we show all 40 results in a random order. In total five experts with different degrees of experience are consulted. This results in $5 * 400 = 2,000$ comparisons. Figure 5.1 shows an example of a master defect and associated result list.



Master defect synopsis: Maecenas id ligula malesuada, luctus sem vitae, venenatis nibh.

Master defect description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu tortor rutrum, adipiscing dui sit amet, auctor massa. Suspendisse quis arcu tincidunt, bibendum lorem non, tempor odio. Nullam at blandit dui, et lobortis risus. In venenatis risus nulla, id cursus.

Result set:

	Defect synopsis
1	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
2	Duis tempor ligula sit amet purus pretium ultricies.
3	Donec in augue varius, porta dui id, dictum urna.
4	Nulla et lorem tincidunt, blandit sapien eu, rutrum est.
5	Nunc eu tortor ullamcorper, congue nisl id, pellentesque sapien.
6	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
7	Duis tempor ligula sit amet purus pretium ultricies.
8	Donec in augue varius, porta dui id, dictum urna.
⋮	⋮
40	Nunc eu tortor ullamcorper, congue nisl id, pellentesque sapien.

Figure 5.1: *Example of a master defect with its result list.*

5.2 Definition of "related defect"

The concept of a 'related defect' can be interpreted in many different ways. To prevent ambiguities, we determined the following definition in co-operation with an expert.

A defect is related when:

- it is a duplicate of the master defect;
- the defects describe the same problem but the products on which the bug occurs differ;
- the defects have similar solutions (e.g. both are deadlocks that occur due to releasing interfaces in the wrong order);
- the defects have similar symptoms (e.g. both defects result in incorrect contradictions in mixed orientation jobs).

5.3 Expert selection

Determining if two defects are related is in some cases quite hard, especially if the problem description of the master defect is not clear to the expert. To reduce this problem as much as possible, we choose to select all master defects from a particular subsystem and select experts working on the same subsystem. However, the result set of all algorithms and random selection use the whole set of defect reports (including all subsystems).

5.4 Algorithms

In recent studies regarding duplicate detection, synopsis (short description) and description are most commonly used for comparing defect reports [RAN07, SLW+10, JW08]. Therefore, we decided also to use these two fields to search for related defect reports. We present three different algorithms to generate a related defect list. The first algorithm compares two defects based on the synopses. The second algorithm compares the descriptions of the defects. The third algorithm compares the defect based on a combination of the synopsis and description. As mentioned before, we also add 10 randomly selected defects.

5.4.1 Algorithm 1 / 2: Comparison based on synopsis / description

Algorithm 1 and 2 are almost the same, except for one variable, therefore we discuss them together in this section.

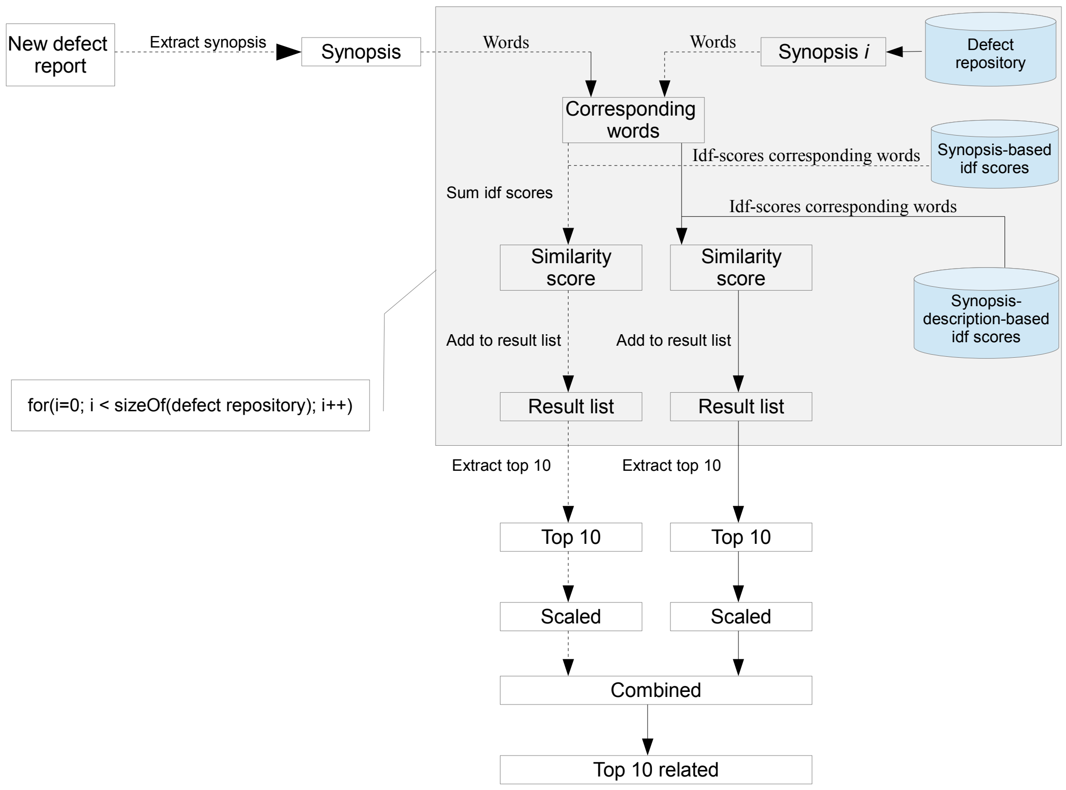


Figure 5.2: *Process of algorithm 1 to generate the top 10 result list. Algorithm 2 can be obtained by replacing the synopsis in the dashed line process by the description and replacing the synopsis-based idf scores by the description-based idf scores.*

Figure 5.2 shows the process of obtaining the result list using Algorithm 1. The process starts by extracting the synopsis from the new defect report and from the defects in the repository. Then the corresponding words between the new defect and the defect in the repository are obtained. These words are preprocessed according to the steps described in Chapter 6. From these words, the associated idf scores are obtained. Idf is a measure which expresses if a term is common (low idf value) or rare (high idf value) across all documents. In Chapter 6 idf is explained in more detail. We distinguish three different kinds of idf scores. One is based on all synopses (*Algorithm 1*), one is based on all descriptions (*Algorithm 2*) and the third one is based on the concatenation of all synopses and descriptions. Taking the sum of idf values of corresponding words results in three similarity scores: synopsis-based idf (*Algorithm 1*), description-based idf (*Algorithm 2*) and synopsis-description-based idf (*Algorithm 1* and *Algorithm 2*).

These scores are computed for all comparisons between the new defect and the defects in the repository. When all scores are computed, the defect reports are sorted on the similarity scores in descending order. Sorting is performed separately for synopsis-based and synopsis-description-based result lists.

Defect	Total score	Scaled
101	6.2745	1
104	6.1949	0.9072
103	6.1494	0.8542
105	6.0220	0.7057
107	5.8931	0.5554
102	5.8201	0.4703
106	5.6787	0.3055
109	5.6787	0.3055
110	5.4166	0
108	5.4166	0

(a) *Top 10 idf scores based on idf-synopsis*

+

Defect	Total score	Scaled
106	3.9404	1
109	3.9404	1
103	3.9305	0.9781
102	3.9226	0.9607
111	3.9074	0.9273
112	3.9074	0.9273
101	3.7867	0.6616
104	3.7799	0.6466
113	3.4861	0
114	3.4861	0

(b) *Top 10 idf scores based on idf-synopsis-description*

=

Defect	Total score	Scaled
103	10.0799	1.8323
101	10.0611	1.6616
104	9.9748	1.5538
102	9.7427	1.4310
106	9.6192	1.3055
109	9.6192	1.3055
111	3.9074	0.9273
112	3.9074	0.9273
105	6.0220	0.7057
107	5.8931	0.5554

(c) *Combined results of idf-synopsis and idf-synopsis-description*

Table 5.1: *Combining two top 10 results (*Algorithm 1*). In red an example is given of combining two results into the combined result.*

As can be seen in Table 5.1a and 5.1b the total scores of the synopsis-based idf (*Algorithm 1*) approach are much higher than the synopsis-description based idf approach. This can be explained by the fact that a description contains in general many more unique terms than a synopsis. Likewise, the chance that a term exists in many documents is a lot higher. Therefore, synopsis-description based idf values are smaller than synopsis based idf values. To prevent this from influencing the results, we scale both result lists to a range between 0 and 1. In Algorithm 2, based on the description, scaling would not have such a big influence because the probability a word shows up in the synopsis but not in the description is much smaller.

Scaling is performed by Equation (5.2) where *minimum* is defined by the minimum top 10 similarity score and *maximum* is defined by the maximum top 10 similarity score.

$$scaled(x) = \frac{x - minimum}{maximum - minimum} \tag{5.2}$$

After scaling, we combine both lists by summing their scaled scores and sort them according to their scaled score. Table 5.1c shows the result of combining two result lists.

5.4.2 Algorithm 3: Comparison based on concatenation of synopsis and description

Algorithm 3 uses the concatenation of the synopsis and description to compare two defects with each other. In contrast to Algorithm 1 and 2, Algorithm 3 only uses the synopsis-description based idf score for computing the similarity score. Therefore, no scaling is needed in this case. The result list is sorted on the sum of idf scores of the corresponding words.

5.4.3 Random selection

To prove that our algorithm performs better than a random algorithm, we add 10 randomly selected defects from the Synergy database. The randomly selected defects may not be the same as the master defect. Furthermore, each randomly selected defect should be unique.

5.5 Evaluating results

After all experts have finished marking the defects, we evaluate the results. During this phase we determine which algorithm performs the best. Furthermore, we check if the algorithms perform better than the random selection.

We also try to improve the best performing algorithm by analyzing the result scores and to look for similarities between the related defects or not related defects.

5.6 Implementation Related Defect Finder (RDF)

As a final step we implemented the best algorithm(s) in a tool called Related Defect Finder (RDF). During this phase we continued applying techniques to improve the search results. We also investigated to improve RDF by processing userdata at runtime.

In the RDF tool we logged all statistics which could be useful to improve the tool in a later stadium.

To prevent the Related Defect Finder tool from being too slow, we require that the implemented algorithms takes on average at most 10 seconds to compute the results.

6 Natural Language Processing Techniques

Bug reports are mostly written in natural language (NL). Before comparing natural language data some pre-processing is needed. For example, we need to determine how to split a sentence into words and how to handle different forms of verbs. As a basis we used the processing stages described by Manning and Schütze [MS99]. We discuss the following processing stages:

- Tokenization
- Stemming
- Stop words removal
- Synonyms replacement
- Spell checking

For each of these processing stages we explain how we applied these processing techniques in detecting related defects. Then four different similarity measures are discusses.

6.1 Tokenization

During tokenization, a stream of text is split up into words, phrases or symbols. During this process also capitals, punctuation, brackets etc. are removed. In general a word is a string of alphanumeric characters surrounded by a white space.

Most of the punctuation marks are easy to remove. For example the comma and semicolon are easy to remove because they are not part of the word. However, punctuation marks like periods, hyphens and apostrophes can be processed in different ways. For example, abbreviations such as e.g. and etc. and the end of sentences are marked by a period. Furthermore, decimal numbers or version numbers also contain periods.

Hyphens should split the word in one case and in the other case it should be kept together. For example *e-mail* should be kept together, while *so-called* could better be split [RAN07].

Thus, splitting text into words is not as trivial as it seems to be. The type of data to tokenize influences the approach of splitting into words.

6.1.1 Tokenization in detecting related defects

For tokenization we used a set of characters as delimiters. These character were determined by analyzing the defect reports. A white space and the following characters are used to split the text:
“ ” ∼ | + \r (carriage return) \n (new line) \t (tab) . : ; ‘ } { " ’ > < ( ) - / % ! # &] [= , \ ? *

The period is also present in this list, but it needs some extra attention. A version number, like 12.6.62.6, is an example of a term in which periods should not be removed. Therefore, the algorithm does not split on periods in floating point numbers. Using a regular expression we can distinguish between (version) numbers and the end of a sentence.

During the tokenization process also all capital letters are transformed into lower-case. Furthermore, we also remove accents, and diaeresis from the following letters: *é, è, ë, ê, ö, ó, à, á, ä, â, ü, ù, ú, ç, ï, î, í*, and replace the *ß* by *ss*.

6.2 Stemming

Words can be written in different kinds of grammatical forms. However, the meaning of the word is the same. The stemming process transforms a word into its ground form by removing affixes and other lexical components. For example, *performance* and *performing* are transformed into the ground form *perform*.

Most commonly [KMM00, JW08] used English stemming algorithm is Porter [Por80]. In 2001 Porter published an improved version of his algorithm: Porter2 [Por01].

Lancaster [Pai90], by Paice and Husk, is another stemming algorithm. Paice/Husk stemmer only evaluates the suffix of each word and uses just one table of rules, while Porter also considers consonants and vowels and six different steps of rules which have to be executed consecutively.

6.2.1 Stemming in detecting related defects

For stemming we used the Porter algorithm. It is widely used [AKAD+13, JW08, MGL09, ZCT+04] and according to Manning and Schütze [MS99] *"it has repeatedly been shown to be empirically very effective"*.

A disadvantage of stemming is the fact that it does not consider irregular verbs. For example, *be*, *been* and *was* are not transformed into the same ground form. A lemmatizer is needed to solve this issue. A disadvantage of a lemmatizer is that it needs a complete vocabulary to correctly transform words. As a result, applying a lemmatizer has a big influence on the performance [MS99]. Therefore, we decided to not apply a lemmatizer.

6.3 Stopword Removal

There are a lot of words which do not carry any specific information and hence are not likely to be useful in similarity analysis. Such words like *the*, *that*, *an* and *a* are called stopwords. Most of them are prepositions, conjunctions or pronouns [RAN07]. Because those words are not significant, they could be removed. If not removed, those stopwords could disturb the similarity calculation. The list of stopwords to use depends on the domain it is applied to [MS99]. Previous studies show that removing stopwords improves the performance [JPPC12, Bra09].

To identify the stopwords instead of using a predefined list, inverse document frequency (explained in 6.6) could be applied [MBK91] . This measure of a word's importance assigns a weight to each term based on the number of documents it appears in. If a term shows up in many documents, it is considered to be less important and thus its weight for the similarity calculation is lower. As a result the stop words have a very small influence on the similarity calculation.

6.3.1 Stopword removal in detecting related defects

The algorithm we applied to calculate the similarity score is based on the sum of all corresponding words within two documents. Therefore, a stopword list is needed to prevent articles and prepositions from influencing the similarity score. Furthermore, not all descriptions are written according to a template (Figure 2.1). If we do not add the words of the template to the stopword list, then descriptions written according to a template would have on average a higher similarity score when comparing to a template based description. All words of the template are thus also added to the stopword list. The complete stop words list can be found in Appendix D.

6.4 Synonyms replacement

Synonyms should also be taken into account when applying nlp. Each person uses his/her own vocabulary. This approach also can improve the similarity results. Howard et al. proposed a method to create semantically-similar word pairs in a domain specific environment [HGPVS13]. In their study they use the leading comment and method signature of a method (in Java) to automatically match two words as being synonyms. As a result an accuracy of 87% has been achieved in identifying descriptive comments. Although the authors achieved a high accuracy, mismatches still can have a major impact.

6.4.1 Synonyms replacement in detecting related defects

We created a synonym list manually for the most frequent words in the defect reports. Applying the approach of Howard was not possible because at Océ there are no strict rules regarding structure of comments.

We decided to take the most frequent word list (top 50) from all defect reports and studied which words were used interchangeably. We did not only considered pure synonyms but also domain specific synonyms. For example, *11504*, an error number, always refers to a crash. Thus, we replaced all occurrences of *11504* by *crash*. Furthermore, a number of UK English words are normalized to US English words (e.g. colour $\rightarrow$ color). During this step we also consider some words which are written in different ways like *scan2mail*, *scantomail* and *scan2email* and translate them into one form. Synonym replacement is executed before stemming. For some verbs for example it could be beneficial to replace the stemmed word by the stem of its synonym (e.g editable $\rightarrow$ changeable). This prevents from adding all forms of a specific verb to the synonym list.

6.5 Spell checking

Misspellings have a great influence on disturbing the similarity results. Van Delden, Bracewell and Gomez [vDBG04] presented two algorithms to automatically improve text quality. However, bug defects contain many domain specific terms which are not recognized by spell checkers. For example, function names could be corrected into different words which will result in more hits. In such a case this behavior is undesirable, because a match on a function name is more preferable.

Some studies analyzed most common spelling mistakes / typos [1] [2] [3]. As a result they came up with a list which contains the most common spelling mistakes. As an alternative such a list can be used. Each misspelling which is present in the list is replaced by its correctly spelled word. Furthermore, this approach prevents the domain specific terms from being corrected automatically. A disadvantage of this approach is the fact that it only treats common misspellings and not domain specific misspellings. Of course it is possible to extend this list, but extending such a list takes a lot of time.

6.5.1 Spell checking in detecting related defects

At Océ for most employees English is not their native language, therefore chances for misspellings are even bigger. To reduce the influence of misspellings we decided to combined three lists of commonly misspelled words [1] [2] [3].

These lists are based on misspellings but also include typos. The combined list contains in total 296 misspellings (Appendix B). A huge disadvantage of this list is the fact that it is not domain specific. Another disadvantage is the fact that this list is based on misspellings made by English-speaking people, whereas most employees are Dutch which could result in different spelling mistakes. A way to improve this list, is to analyze all terms manually and check for most common misspellings and typos. Unfortunately this approach would take a lot of time. Indeed, approximately 90,000 unique words have to be evaluated manually.

Applying an automatic spell checker was not an option because we have to deal with a domain specific language.

6.6 Similarity measures

Cosine, Dice and Jaccard are three most common measures for calculating the similarity between two documents. BM25 is another well known ranking function. In contrast Cosine, Dice and Jaccard, BM25 depends on two parameters which can be used to optimize the function. For calculating the similarity between two documents, some similarity algorithms need a vector array as input. We discuss three different techniques to calculate the vector array first.

Vector array

Most used techniques are term frequency (tf) [RAN07], inverse document frequency (idf) [SLW$^+$10] and term frequency - inverse document frequency (tf-idf) [WZX$^+$08].

Term frequency (equation 6.1) calculates the number of times term t occurs in document d denoted by $f(t,d)$, divided by the sum of all frequencies, denoted by $\sum\limits_{w\in d} f(w,d)$.

$$\text{tf}(t,d) = \frac{f(t,d)}{\sum\limits_{w\in d} f(w,d)} \tag{6.1}$$

The inverse document frequency (equation 6.2) expresses whether term t is common or rare across the set of all documents D. It is obtained by dividing the total number of documents by the number of documents containing term t. Taking the logarithm of the quotient results in the idf value. To prevent dividing by zero, we add a one to the denominator. A division by zero would occur if term t does not appear in the set of documents D at all.

$$\text{idf}(t,D) = \log \frac{|D|}{1 + |\{d \in D : t \in d\}|} \tag{6.2}$$

The third technique we discuss is tf-idf (equation 6.3) which is the product of term frequency and inverse document frequency and expresses the importance of a term to a document in a collection of documents. A term appearing frequently in a document but is rare across the set off all document results in a high tf-idf value.

$$\text{tf-idf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D) \tag{6.3}$$

6.6.1 Cosine similarity

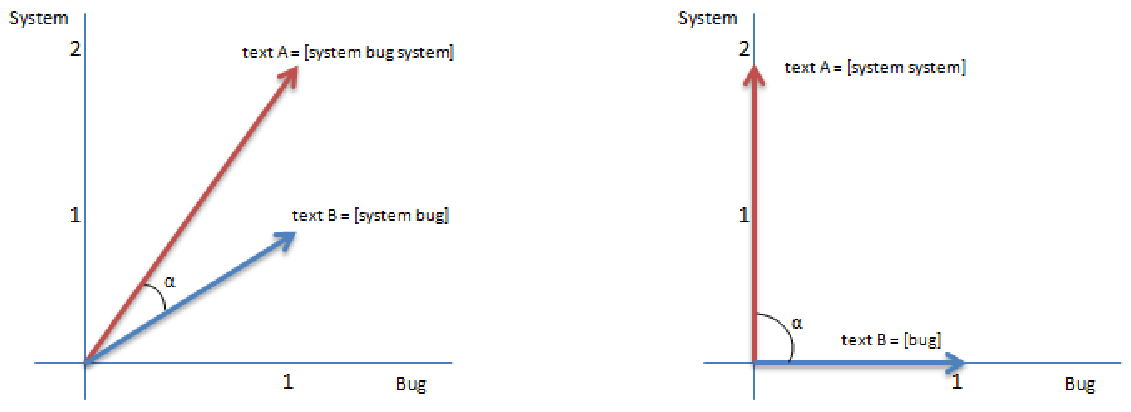
Cosine Similarity [JW08, RAN07] is a measure of similarity that calculates the cosine of the angle between two vectors. If two vectors are equal, the angle between the two vectors is 0° and the Cosine similarity is 1. If the similarity between the vectors is 0, then the angle between the two vectors is 90° (Figure 6.1b). Given two vectors A and B, the Cosine similarity can be computed by the equation given in (6.4). Before comparing two texts using the Cosine similarity a vector array has to be computed, for example using tf-idf (6.3).

[1]AskOxford: Commonly Misspelled Words, Oxford University Press, 2009, webpage: http://oxforddictionaries.com/words/common-misspellings

[2]100 Most Often Misspelled Words in English, LoveToKnow, Corp., December 2008, webpage: http://grammar.yourdictionary.com/spelling-and-word-lists/misspelled.html

[3]Canadian, British and American Spelling: Some Commonly Misspelled Words", LukeMastin.com, December 2008, webpage:http://www.merriam-webster.com/dictionary/concensus

$$\text{CosSim}(A,B) = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{6.4}$$



(a) *Representation of the similarity between "system bug system" and "system bug". α-value* $\approx 18.19°$*,* $\cos(\alpha) \approx 0.95$

(b) *Representation of the similarity between "system system" and "bug". α-value* $= 90°$*,* $\cos(\alpha)$ *= 0*

Figure 6.1: *Computations of the cosine similarity. The vector array is based on the number of term occurrences.* $\cos(\alpha)$*=sim(text A, text B)*

6.6.2 Dice coefficient

The Dice coefficient [RRGACD12, LT11], also known as Sørensen index, is another metric to compare two sets of data. In (6.5) the equation to calculate the Dice coefficient between two texts A and B is given. In contrast to the cosine similarity, the Dice coefficient can be applied to a set of words; there is no need to compute a vector array first.

$$\text{DiceSim}(A,B) = \frac{2|A \cap B|}{|A| + |B|} \tag{6.5}$$

For example, if we have text "system bug report" and "system bug", then we get two sets: $A = \{system, bug, report\}$, $B = \{system, bug\}$. The Dice coefficient is then computed as follows:

$$\begin{aligned}\text{DiceSim}(A,B) &= \frac{2|A \cap B|}{|A| + |B|} \\ &= \frac{2|\{system, bug\}|}{|\{system, bug, report\}| + |\{system, bug\}|} \\ &= \frac{2 \times 2}{3+2} \\ &= \frac{4}{5} \\ &= 0.8 \end{aligned} \tag{6.6}$$

6.6.3 Jaccard similarity coefficient

The Jaccard coefficient is also used for comparing the similarity between two sets. This similarity index [NC08, SAMO11] is quite straightforward, it is defined as the size of the intersection divided by the sum of the size of both sets minus the size of the intersection. The equation is stated in (6.7).

$$\text{JacSim}(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{6.7}$$

An example of applying the Jaccard similarity coefficient is shown in equation (6.8).

According to texts "system bug report" and "system bug" the associated sets are: $A = \{\text{system, bug, report}\}$, $B = \{\text{system, bug}\}$.

$$\begin{aligned}
\text{JacSim}(A, B) &= \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \\
&= \frac{|\{system, bug\}|}{|\{system, bug, report\}| + |\{system, bug\}| - |\{system, bug\}|} \\
&= \frac{2}{3} \\
&\approx 0.67
\end{aligned} \tag{6.8}$$

6.6.4 Okapi BM25

BM25 [SLKJ11, RZT04, ZCT+04, AKAD+13] is a popular ranking function based on the probabilistic retrieval framework. It ranks a set of documents using a query containing a bag of words.

Given a document D and a set of search keywords Q, the BM25$_{Score}$ can be calculated using equation (6.9). The inverse document frequency as given in Equation (6.2) is denoted by 'idf'. The term frequency of q_i in document D is denoted by $f(q_i, D)$. The length of document D in words is denoted by $|D|$. dl_{avg} is the average document length over the set of documents of which D is part of.

BM25 contains two parameters: k_1 and b. Parameter k_1 calibrates the document term scaling, using b the scaling by document length can be determined. Scaling helps preventing tiny documents getting ridiculously high weights in $\frac{|D|}{dl_{avg}}$. According to Manning, Raghavan and Schütze [MRS08] a value between 1.2 and 2 is reasonable for parameter k_1. For parameter b, 0.75 is a reasonable value.

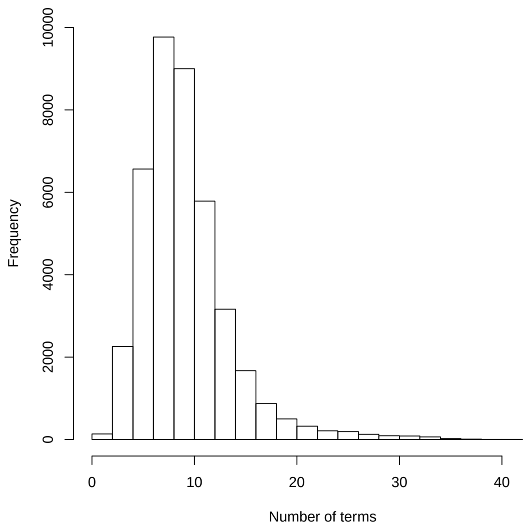
$$\text{BM25}_{Score}(D, Q) = \sum_{i=1, q_i \in Q}^{n} \text{idf}(q_i) \times \frac{f(q_i, D) \times (k_1 + 1)}{f(q_i, D) + k_1 \times (1 - b + b \times \frac{|D|}{dl_{avg}})} \tag{6.9}$$

Similarity measures in detecting related defects

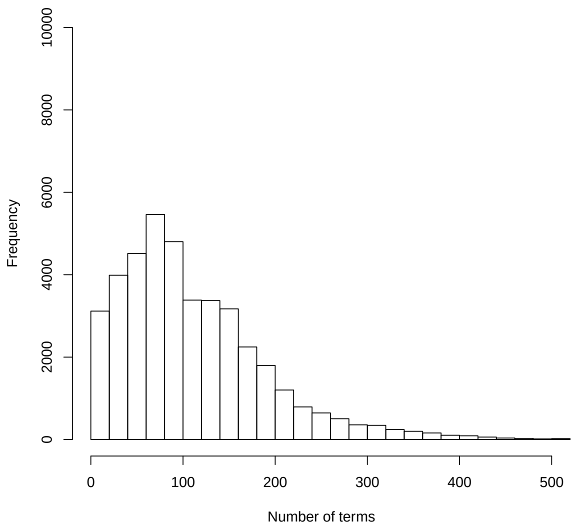
Cosine, Dice and Jaccard are well-known similarity measures. All three algorithms use the document length to determine the similarity coefficient. However, defect report descriptions and synopsis's could differ a lot in length (Figure 6.2). One could add for example an execution log to the description which could have a huge effect on the similarity results. Consequently, we do not apply Dice, Jaccard or Cosine.

BM25 is also used quite often, especially for search tasks [MRS08]. The parameters, which are free to configure, made us decide to not use the BM25 function, because a training set approach is not feasible in our case. That makes it hard to determine the initial parameters. Calibrating the parameters will be even harder.

We decided to follow Sun et al.[SLW+10] and apply the formula shown in equation (6.10). Similarity $\text{sim}(D_1, D_2)$ returns the similarity between two sets of words. The similarity is computed by taking the sum over idf values of all terms in the intersection of set D_1 and D_2. The sum over the empty set is defined as zero.



(a) *Number of terms in the synopsis.*



(b) *Number of terms in the description.*

Figure 6.2: *Number of terms in the synopses and descriptions of a defect report.*

Table 6.1: *Example of taking the intersection of two document sets after applying tokenization, stop word removal and stemming.*

Stemmed set of words in document A:	{dvd, play, return, error, 54320, after, press, button}
Stemmed set of words in document B:	{play, dvd, crash, error, 54320, show}
Intersection:	{dvd, play, 54320}

$$\text{sim}(D_1, D_2) = \sum_{t \in D_1 \cap D_2} \text{idf}(t) \tag{6.10}$$

By taking the sum we do not distinguish between different document lengths. In Table 6.1 an example is given of the intersection between two documents.

7 Defect Relationship Allocator (DRA)

7.1 Introduction

The Defect Relationship Allocator is developed for a group of five selected experts to indicate whether defects are related. Three different algorithms (Section 5.4) are implemented in this tool to test its quality performance. Furthermore, a random selection of defects is added to prove that the algorithms perform better than a random selection. The tool returns at most 40 results (10 per algorithm) for a given defect. If a result is part of different algorithm result lists, it is only added once to the combined result list. Therefore, sometimes less than 40 results are displayed.

7.2 Retrieving data

IBM Rational Synergy (IRS) is used as Océ's software configuration management (SCM) system. All change requests, both features and defects, are stored into this system. The built-in query language to retrieve the data from IRS is very limited. Well known functions like `SUM`, `COUNT` and `GROUP BY` are not supported. Furthermore, the IRS database has some performance issues when it comes to requests with a large result. Therefore, we decided to setup our own database and copy all defects (till 07-05-2013) once. Newly submitted defects are thus not added to our database.

7.3 Processing data

Listing 7.1 shows the query extracting all defect information from the Synergy database. A major disadvantage of the IRS query language is the fact that it is not possible to limit the number of results. As a consequence the query returns a very big data set which ultimately results in a memory overflow on a local machine with 4GB of RAM.

After analyzing the defects it seemed that both *problem description* and *transition log* contain a lot of data per defect. Therefore, we decided to split the data retrieving process into two steps. We first collected all fields except the problem description and transition log. Then we iterate over all defect to collect the problem description and transition log one by one. Using this approach we prevent the system from throwing a memory overflow exception. The commands to retrieve the *description* and *transition log* are shown in Listings 7.2 and 7.3, respectively.

Parameter *-f* allows the user to select the fields returned by the query. Because the query result is returned as plain text, we have to add a unique separation stabbing to distinguish between the different fields and rows. Default operators such as '|' are sometimes used in descriptions, therefore we decided to create unique separation stabbings: *#Y#Y#* distinguishes property fields and *#Q#Q#* distinguishes between defect entries.

Listing 7.1: *Command to retrieve all defects from IRS*

```
         query "(request_type='Defect') and
                (cvtype='problem')"
        -f
        "#Y#Y# %problem_number #Y#Y#
        %change_request_synopsis #Y#Y#
        %problem_description_OFF #Y#Y#
        %request_type #Y#Y#
        [...]
        %owner #Y#Y# #Q#Q#
```

Listing 7.2: *Command to retrieve the description of defect 've070#12345'*

```
cr('ve070#12345') -f "#Y#Y# %problem_description #Y#Y# #Q#Q#"
```

Listing 7.3: *Command to retrieve the transisition log of defect 've070#12345'*

```
cr('ve070#12345') -f "#Y#Y# %transistion_log #Y#Y# #Q#Q#"
```

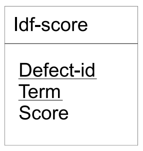
7.4 Caching data

All three algorithms use an idf score to compute the result list. The algorithms use three different kinds of idf scores: synopsis-based, description-based and synopsis-description-based. Computing the idf score over approximately $40,000$ defects at real time is a heavy calculation and thus, very time consuming. Indeed, every synopsis and description has to be checked for each of the words in the given defect. All algorithms have to compute their results within a couple of seconds to achieve an acceptable user experience (Section 5.5).

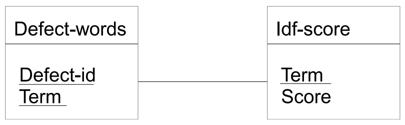
To improve the performance, we decided to cache all three different idf scores in a database. An advantage of using idf in contrast to tf-idf is that each word only has one score for the entire dataset rather than one score per document, which reduces the number of records and makes updating the scores a lot faster.

In the idf score database table the *defect id* (primary key), *word* (primary key) and *idf score* are stored (Figure 8.1a). We created three tables according to this design: for synopsis-based idf scores, for description-based idf scores and for synopsis-description-based idf scores. All words are stored after applying the techniques discussed in Chapter 6. For each unique (word, defect id) combination we store the associated idf score. Using this design we can easily find defect reports in which a particular word is present.

According to database normalization rules we should have created two tables (Figure 8.1b). The approach we used results in a lot of redundant idf-scores, indeed each unique term is associated to one idf-score. For simplicity reasons we decided to keep it like this for the DRA tool. In the subsequent RDF tool we do split up the two tables according to the database normalization rules.



(a) *Database design as implemented in DRA.*



(b) *Database design according to normalization rules.*

Figure 7.1: *Two database table designs to store the idf scores. In 8.1a the implemented database design. In 8.1b the normalized variant.*

By executing the second query in Appendix C related defects with their corresponding idf scores can be found. The advantage of indexing is that we do not need to compute the idf score on request anymore, which saves a lot of computing time.

7.5 Computing result list

We tested two different SQL queries to compute the result list, to observe the difference in performance. The first query was based on an *EXISTS* statement, while the second approach only used *INNER JOIN* statements.

To test the performance we executed the query for 10 different defects such that we could prevent the database from caching. As a result the second approach (0.2 seconds) performed approximately 51 times better than the first approach (12.6 seconds). Both queries can be founds in Appendix C.

7.6 User interface

DRA is developed as a web-based application such that an installation is not required. Before getting started the user has to login using his/her Océ username and the number of months of experience with the particular subsystem *S*. After logging in, the screen, which is shown in Figure 7.2, is displayed. In the upper right corner, the user name and the number of months of experience with the subsystem are displayed.

On top of the application a random master defect is given. The DRA tool shows the synopsis and description of the master defect such that the expert has a clear view of the problem. The master defect is randomly chosen from all defects within the subsystem *S* each time the page is loaded. To the left of the master defect two icons are shown. By clicking on the refresh icon (), the page refreshes and a new master defect is selected randomly. Clicking on the IBM Synergy logo () opens the defect report of the master defect in Synergy.

On the lower part of the page the result list according to the three algorithms and the random selection is shown. For each of the "related" defects the synopsis is shown. On the left of each resulting defect three icons are given. By clicking on the thumbs up icon (), the defect can be marked as related. The thumb then becomes green. Likewise, clicking on the thumbs down icon () marks the defect as "not related". In this case, the thumb becomes red. We call this process voting.

Again by clicking on the Synergy logo (), the defect is opened in Synergy to provide more defect information, such as description, transition log and product name.

After marking all defects, refreshing the page shows a new master defect on top and a new result list is computed.

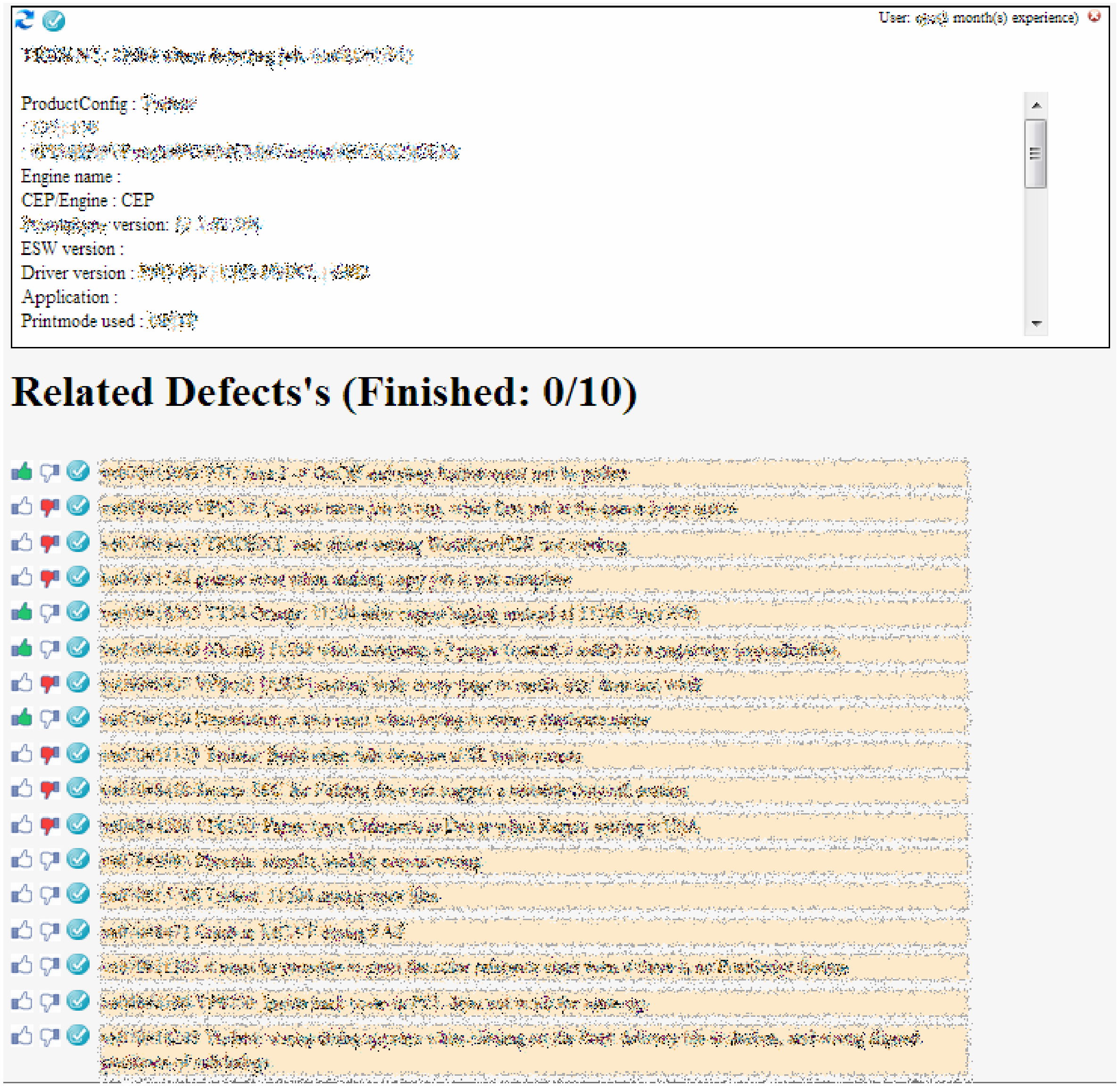
7.7 Database setup

All defects are stored in a MySQL database. For each defect we store all fields extracted from the Synergy database using the command in Listing 7.1 supplemented with the description and transition log.

Votes (related / not related) are also stored into the database. For each vote we store the master defect id, the marked defect id, the vote(related / not related), the current time stamp, the expert's user name, the algorithm from which the marked defect was computed and the position in the algorithm result (between 1 and 10). A list of all stored fields is given in Table 7.1.

Vote
Id
MasterDefect
Defect
IsRelated
Timestamp
Username_expert
Algorithm
ResultPosition

Table 7.1: *Vote table of the DRA system. This table contains information about the relationship between two defects.*



User: [redacted] month(s) experience)

[redacted]

ProductConfig : [redacted]
[redacted]
[redacted]
Engine name :
CEP/Engine : CEP
[redacted] version: [redacted]
ESW version :
Driver version : [redacted]
Application :
Printmode used : [redacted]

Related Defects's (Finished: 0/10)

[redacted]

Figure 7.2: *Example of the DRA user interface. Related defects are marked by a green thumb up icon. A red thumb down icon tells that the associated defect is not related.*

Furthermore we store the number of months of experience with the subsystem for each of the users. For each master defect we store the performance of computing the results in milliseconds which comprises the sum of computation time of all algorithms.

7.8 Results

In terms of correctness, all three algorithms performed better than the random selection, which was expected. Overall, the algorithm based on the synopsis performed the best. In 41.7%, the result, which was returned on the first place, was marked as related. The description based search (29.7%) and the synopsis-description based search (25.7%) also performed relatively good for the best ranking place. For the random selection, none of the defects which appeared on the first place was related.

In Figure 7.3 the percentages of related defects per position are given for each of the algorithms. The performance of the description based and synopsis-description based algorithms is comparable. The synopsis-based algorithm outperforms these two algorithms in all cases but one. If we take a 10 percent error margin into account, a slightly decreasing trend can be observed.

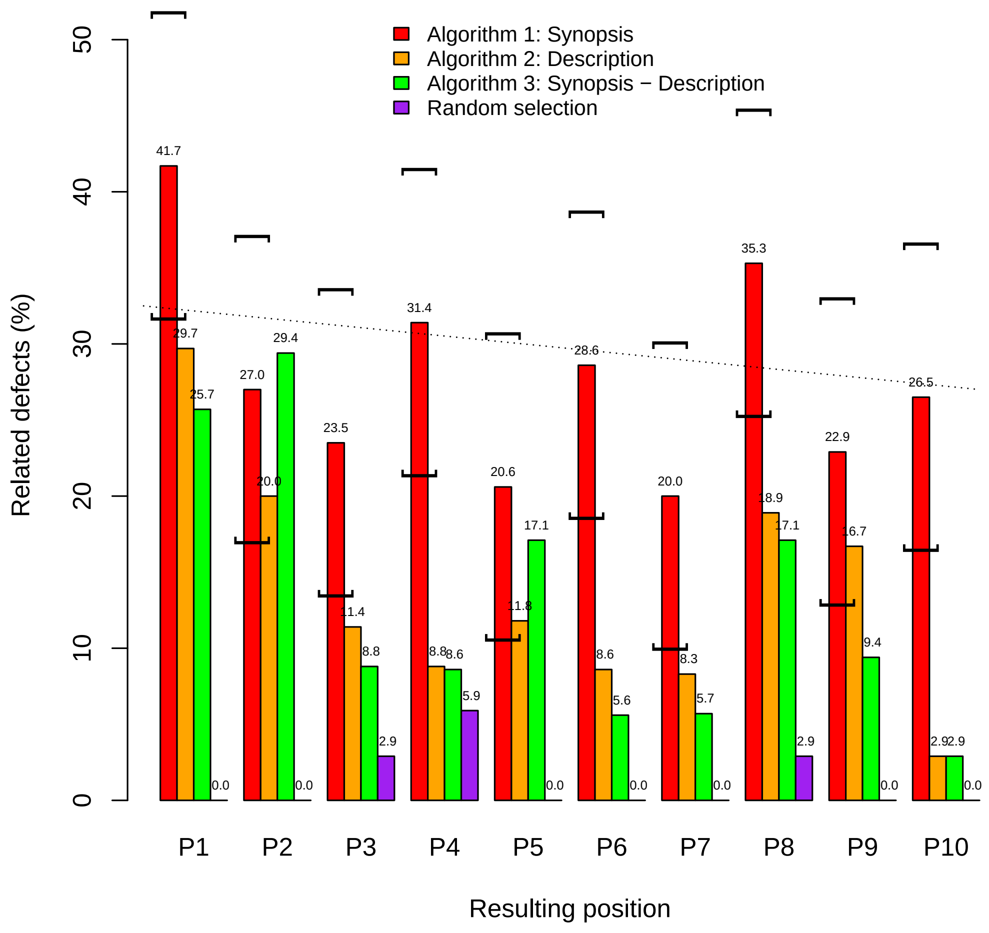


Figure 7.3: *Percentage of related defects, per algorithm, per position. For Algorithm 1, 10% error margins are given by vertical brackets. The dotted line shows the decreasing trend of algorithm 1 which could be observed with respect to the 10% error margins.*

The random selection performs, as expected, in all cases the worst. For all random results only 1.15% was marked as related. In case of the synopsis-based algorithm 27.8% of all votes was marked as related. The description-based algorithm was marked as related in 13.9% of all results. For the synopsis-description-based algorithm 13.1% of all results was marked as related. Figure 7.4 shows the results of the probability that at least one defect in the top 10 is related. Algorithm 1 (synopsis-based) performs the best. In more than three-quarter of the result lists at least one related defect showed up in the result list. Algorithm 2 (description-based) and Algorithm 3 (synopsis-description-based) perform quite the same. The random selection performs the worst. In less than 10 percent of the result list a related defect showed up.

7.8.1 Overlapping results

In Figure 7.5 the overlapped results between the different algorithms are shown. A result is overlapped between two algorithms if it shows up in both result lists, whether it is related or not. From the figure we can conclude that the overlap between the synopsis-based and description-based algorithm is relatively small. Only 2.33% of the results overlap. Therefore, combining those two algorithms could, theoretically, result in a massive

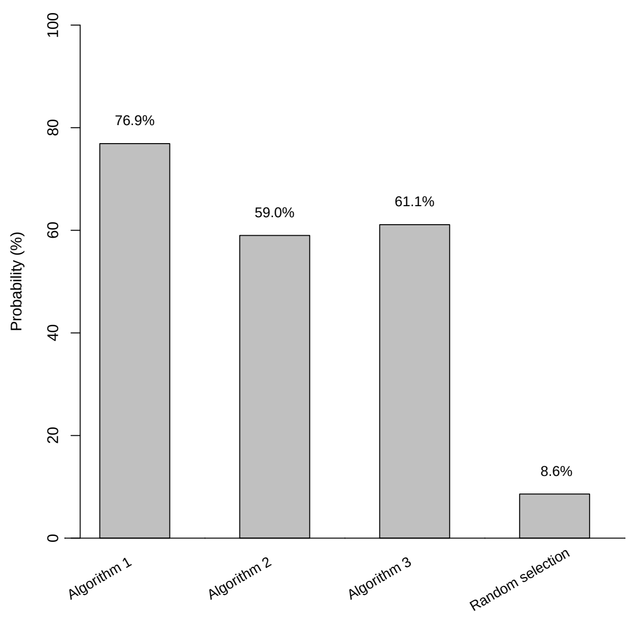


Figure 7.4: *Probabilities that at least one defect in the top 10 is related.*

improvement. Another observation is the fact Algorithm 2 and Algorithm 3 have a very large overlap between their results. From this we can conclude that the synopsis has a very small influence on the result when it is combined with the description. This is not very surprising, because on average the description is approximately 911 characters long where the synopsis has an average length of only 62 characters, which means only 6.4% of the combined text belongs to the synopsis. The large overlap between Algorithm 2 and Algorithm 3 results in the fact that the overlap between Algorithm 1 and Algorithm 3 is also relatively small.

7.8.2 Results per expert

All experts have varying degrees of experience with the subsystem. This could also influence the voting process. In this subsection we discuss the relation between the results of the voting process and the number of months of experience.

In Figure 7.6 the precision for each of the experts per algorithm is given. From this figure we can conclude that in general experts with less experience (Expert 1) marked defect reports as relevant more often than an expert with a lot of experience (Expert 5).

Another interesting observation is the fact that Expert 3 and Expert 5 have equally distributed results for Algorithm 1, 2 and 3. In case of Experts 1, 2, and 4 Algorithm 1 performed much better than Algorithm 2 and 3. In comparison, the differences between the various algorithms for these 3 experts are the same; Algorithm 1 performs approximately 20 percent better than Algorithm 2 and 3, while Algorithm 2 and 3 perform roughly the same.

Furthermore, for Experts 2 to 5 Algorithm 2 and 3 perform quite the same (approximately between 8% and 18%). In Algorithm 1, the differences are much greater. This could be due to the fact that at first sight for a less experienced expert many more defects seemed to be useful than for an experienced expert who already has this knowledge. It could also be that more experienced experts are better in indicating whether a defect is relevant which means less experienced experts are too optimistic.

The only exception in the figure is Expert 4. However, after an interview it seemed that Expert 4 also marked defects as related when he thought it could be useful for less experienced experts but not for himself. Therefore, we consider Expert 4 as being a less experienced expert.

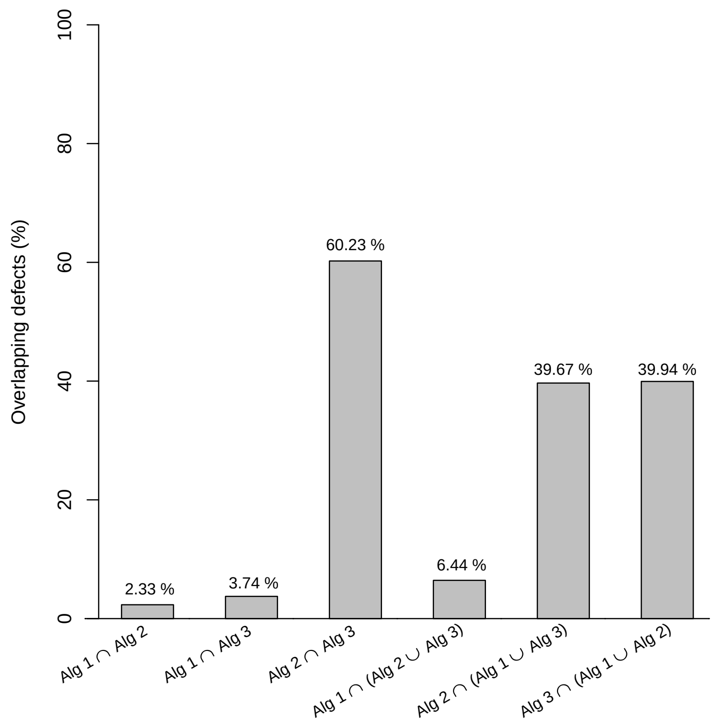


Figure 7.5: *Independences between the different algorithms.*
Alg 1: synopsis-based, Alg 2: description-based, Alg 3: synopsis-description-based

Voting times

We also kept track of the times between each vote. We expected that less experienced experts need on average more time than experienced experts to vote.

In Figure 7.7 the experts' vote times are presented. According to the histograms, the two most experienced experts marked most of the defects within 10 seconds. Compared to Expert 2, Expert 4 and Expert 5, Expert 1 and Expert 3 needed in many more cases more time than 10 seconds to vote. Expert 2 is in between those two couples.

In this figure we focus on the first 60 seconds, because most of the votes were completed within this period. However a number of votes also took more time than one minute. This could be explained by the fact that the expert was not able to determine if the defect was related by only viewing the synopsis. Another reason could be that sometimes other tasks where performed during voting, such as taking a (coffee) break.

In Figure 7.8 vote times for each of the experts are presented which take less than 3.5 minutes. We also removed the first vote of all master defects, because calculating the result list and understanding the master defect is also included in this vote time. Each graph is divided into a number of parts. Each part contains the voting times for a unique master defect. Only Expert 5 was able to complete all 10 master defects, therefore the number of parts differs per expert.

Our first observation is that the graph of Expert 4 is the only one which does not contain any big outliers compared to the other four experts. This could mean that Expert 4 only took some breaks after finishing all votes of a particular master defect.

Another observation is that for some defect reports it was probably more difficult to determine if they were related to the master defect. The second block of Expert 5 is a good example of this behavior. The time needed for each vote in this block is on average relatively higher compared to the other blocks.

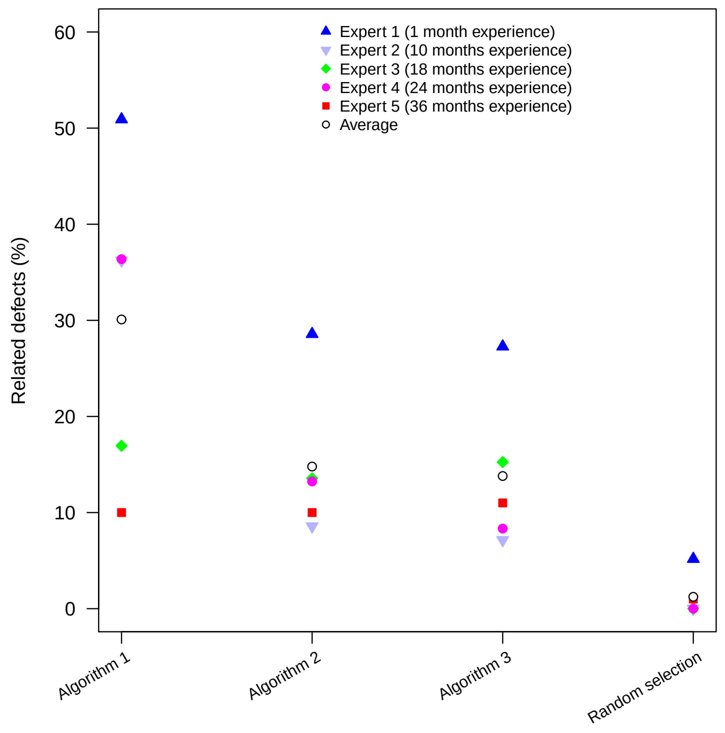


Figure 7.6: *Percentage of related defects per algorithm for each of the individual experts. Algorithm 1: synopsis-based, Algorithm 2: description-based, Algorithm 3: synopsis-description-based*

The high peaks may be caused by the taking of a break by the expert. However, it is still possible that the expert needed in some cases many more time to determine if the defect was related. Another reason for these outliers could be the interruption by a colleague asking a question.

7.8.3 Feasibility

In the DRA tool performance is not an issue as for each of the experts a maximum of 10 result lists are computed. We did however measured the performance of the result list computation such that we can determine if the algorithms are fast enough to use in the subsequent RDF tool. The performance results are shown in Figure 7.9. From this histogram we can conclude that most of the similarity calculations took less than 12 seconds. In these 12 seconds three different result sets are computed, ignoring the time for selecting 10 defect at random. Given these statistics, it is very likely that all of the algorithms meet our requirement of 10 seconds5 for computing the result list.

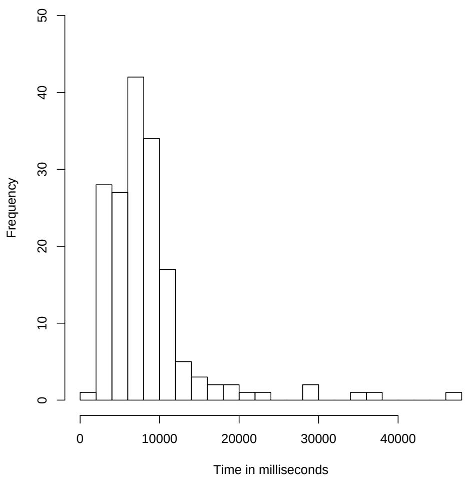


Figure 7.9: *Performance results of computing the result list based on 3 algorithms and 1 random selection.*

7.9 Threats to validity

Voting was performed by five different experts. These experts all have their own interpretation of the notion of "related defect". To minimize the difference in interpretation we wrote a user manual which can be found in Appendix A. The difference in experience also influences voting decisions. We could have let each of the experts evaluate the same list of master defects, to increase the reliability of the results. However, we decided to let each of the experts evaluate a unique list, such that we could gather more data. Another reason for this choice is the fact that difference in experience also result in different knowledge and thus different interpretation. Therefore, some results are useful for less experienced experts but are not for more experienced experts.

For each of the master defects at most 40 defects had to be assessed. As a consequence fatigue or loss of concentration may play a role. As we did not observe the experts during voting, it is hard to say if some breaks where taken. From the voting times we cannot conclude breaks because in some cases it took an expert minutes to find out if the defect was related and in other cases the expert was interrupted by a colleague asking a question.

Voting was performed by only five different experts who resulted in a relatively small set. As a consequence, exceptions have a greater impact on the results. Furthermore we only evaluated one subsystem. Defect reports belonging to other subsystems could contain a more detailed synopsis or description which can have an influence on the results.

7.10 Conclusion

As expected, all algorithms perform, in terms of correctness, much better than the random selection. We did not measured the performance results in terms of speed. However, analysis showed that all algorithms most likely meet the 10 seconds calculation limit.

The synopsis-based algorithm performed the best and therefore is implemented in the subsequent RDF tool. Because the overlap between the synopsis-based algorithm and description-based algorithm was very small, theoretically a significant improvement is possible by combining these two algorithms. The synopsis-description based algorithm performed quite the same as the description-based algorithm. The synopsis-description-based algorithm is a little bit more complicated, therefore the description-based algorithm was selected to use for an optional optimization.

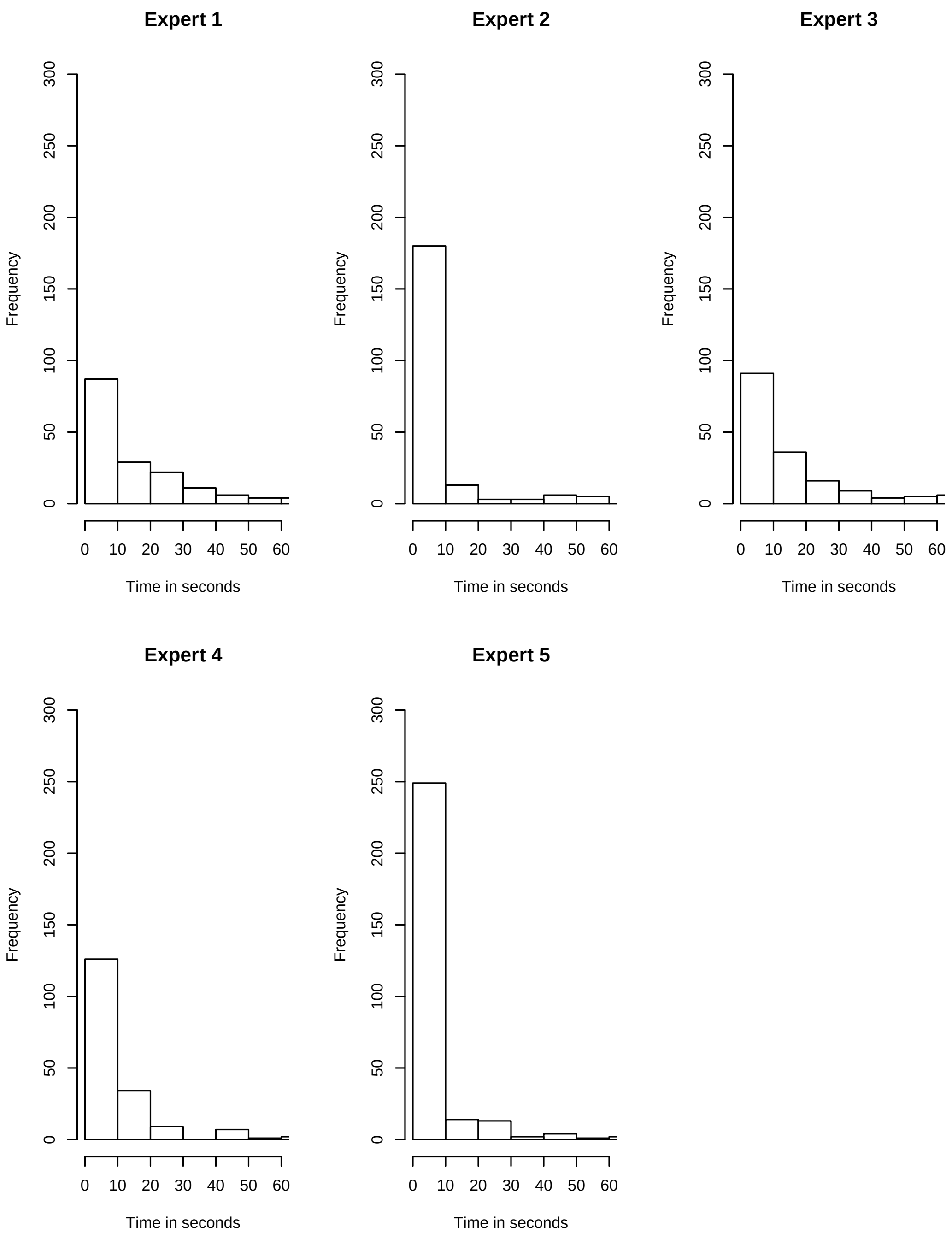


Figure 7.7: *Histogram which expresses the time needed to cast a vote. Expert 1 has the least experience, Expert 5 the most.*

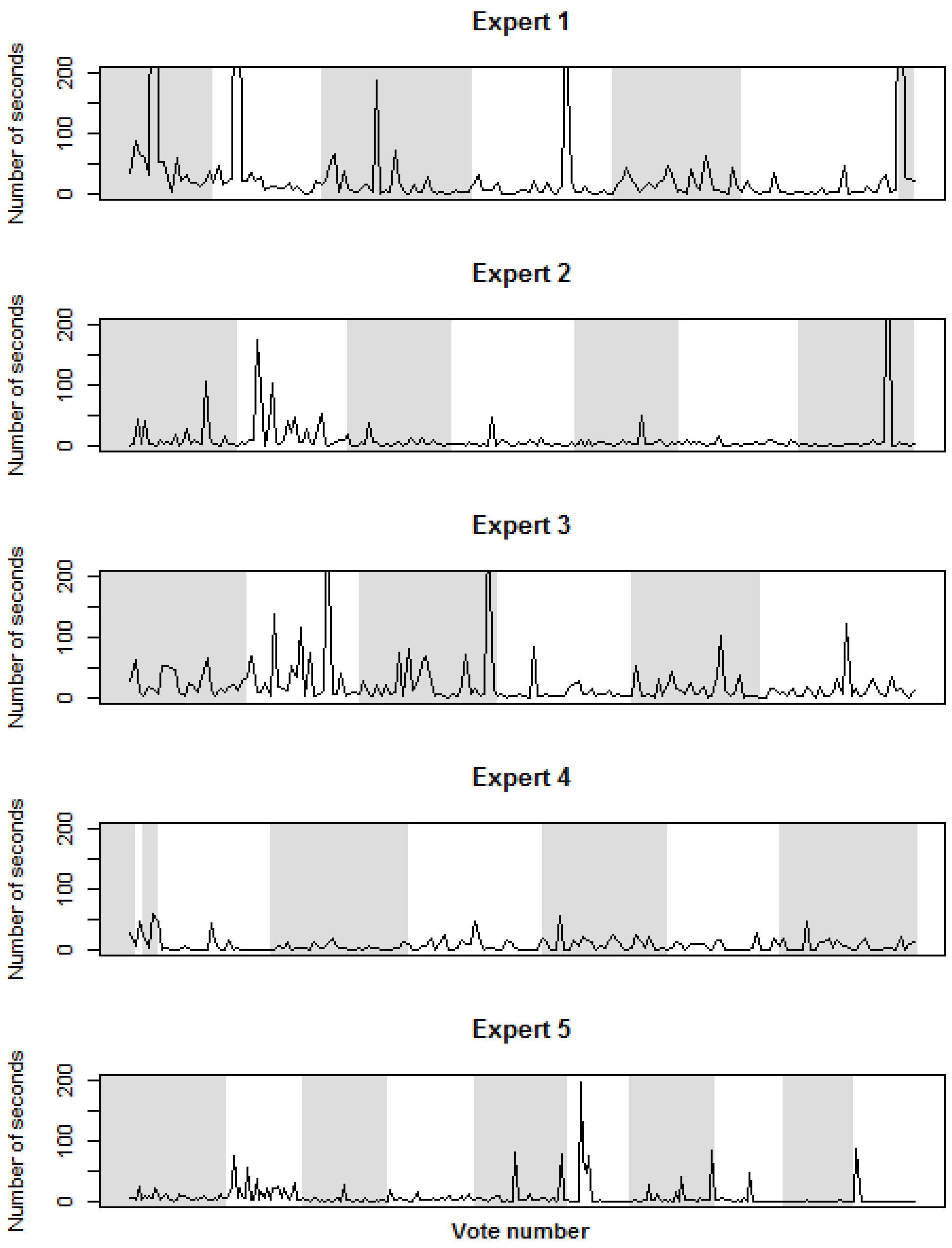


Figure 7.8: *All times needed to cast a vote.*
Expert 1 has the least experience, Expert 5 the most. Each block contains the voting times of the associated master defect.

8 Optimizations

From the previous chapter we conjectured that a significant improvement should be possible by combining the results of the synopsis-based algorithm and the description-based algorithm. In this chapter we discuss different approaches to combine the results of the two algorithms. We used the data that we gathered using the DRA tool to test the suggested optimizations. In the following sections we discuss five different attempts. To improve the reliability, all percentages of related defects are based on at least 10 results.

8.1 Attempt 1: Correlation between a related defect and its similarity score

In this first attempt we investigate if a result is related on basis of its similarity score. In Figure 8.1 we plotted the similarity score of the synopsis-based algorithm and the description-based algorithm with their associated percentage of related defects. The similarity score on the x-axis should be read as equal or greater than. For instance, if the similarity score of a result of the synopsis-based algorithm is equal or greater than five, then the probability that it is related is approximately 28%.

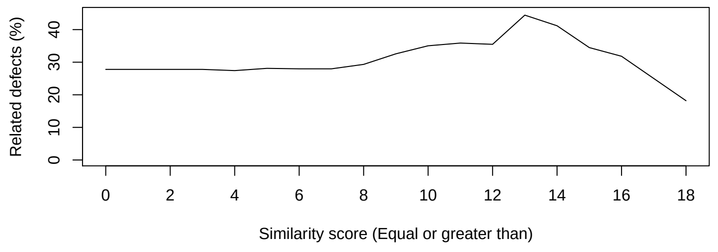
In the synopsis-based algorithm the probability for the defect being related increases till a score of circa 13. Then the graph decreases. If the score is greater than 16 the probability that the defect is related is even lower than in the beginning. This could be explained by the fact that some defects contain very generic synopsis's such as '11504 during shutdown'. The cause of such problems could be anything, therefore a relatively small number of defects would be marked as related in this case.

The relation between the similarity score and the probability that it is related for the description-based algorithm increases till a score of approximately 120. Then it decreases very fast and increases again. Analyzing the associated defect reports did not result in a clear explanation for this phenomenon.

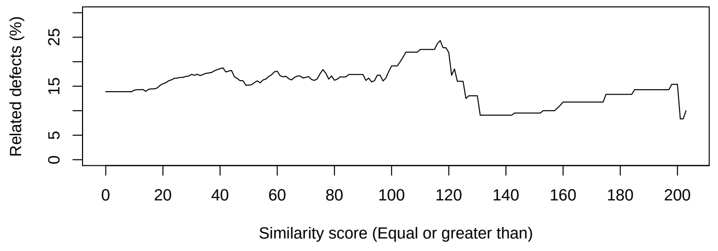
We use the synopsis-based algorithm as a basis. A score of the synopsis-based algorithm which is greater than 16 is replaced by the greatest score of the description-based algorithm, which is equal or smaller than 120. If a score of the synopsis-based algorithm is greater than 16 and in the description-based algorithm all scores are above 120, then no actions has to be taken.

Results

After applying this optimization to the results, it seemed that for this data set the quality performance increased a little bit. Only for a few master defects the result list was improved by applying the optimized algorithm. For some master defects the result list deteriorated. On average 2.71 out of 10 defects are related in the optimized version, which is an increase of 0.7%.



(a) *Synopsis*



(b) *Description*

Figure 8.1: *Percentage of related defects per similarity score. The x-axis should be read as equal or greater than.*

8.2 Attempt 2: Correlation between a related defect and its scaled score

In this second attempt we investigate if there exists a relation between the scaled score as explained in Section 5.4.1 and the percentage of related defects. In Figure 8.2 the probability that a defect is related is given according to the scaled score. Our first observation is that the synopsis-based scaled score shows an increasing trend. Furthermore, the probabilities of the synopsis-based algorithm are overall higher than the probabilities of the description-based algorithm. However, if the description scaled score is higher than 1.65 the probability the defect is related is higher than a score of 0 in the synopsis-based algorithm.

Only a scaled score between 1.0 and 1.4 results in a relatively low probability. However, this could be a small error because the data set was limited. Therefore, we decided to remove the lowest scores if the description-based algorithm scaled score is above 1.65. Because the best result of the description-based algorithm has always a score of 2, the best result is always added to the improved result list. As a consequence, the synopsis-based algorithm result with a scaled score of 0 is removed in all cases.

Results

After applying the proposed modification to the algorithm, the average number of related results per defect decreased by 0.34% to 2.606 out of 10. As a result, this modification is an impairment compared to the synopsis-based algorithm. This impairment is due the fact that a number of related defects were replaced by description-based results which were not related.

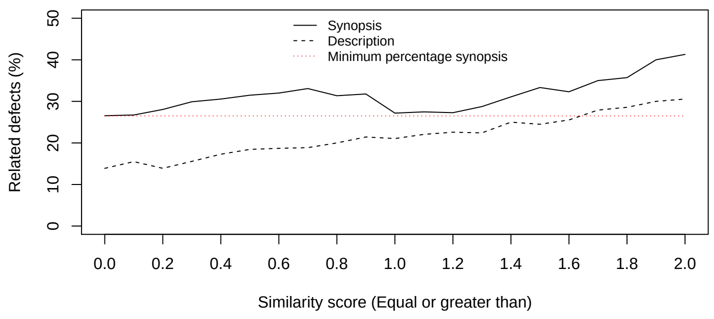


Figure 8.2: *Percentage of related defects per score. The x-axis should be read as equal or greater than.*

8.3 Attempt 3: Correlation between related defect and its position

In this third attempt we investigate if a relationship between a related defect and its position can be discovered. In Figure 8.3 the percentage of related defects per position is shown for both, the synopsis-based algorithm and the description-based algorithm. The idea in this attempt is to combine the percentages of synopsis and description and sort them in descending order. The result of this sorting is shown in Figure 8.4. Only the percentage of the best result of the description-based algorithm was high enough to enter the new optimized result list. As a consequence the seventh result of the synopsis-based algorithm should be removed from the result list.

Results

In summary, we should replace the result on position seven of the synopsis-based algorithm by the best result of the description-based algorithm. However, the differences between the results per position are relatively small. Therefore, it is meaningless to compute the influence of this attempt. Applying this attempt to the complete set of defects would probably return a complete different outcome.

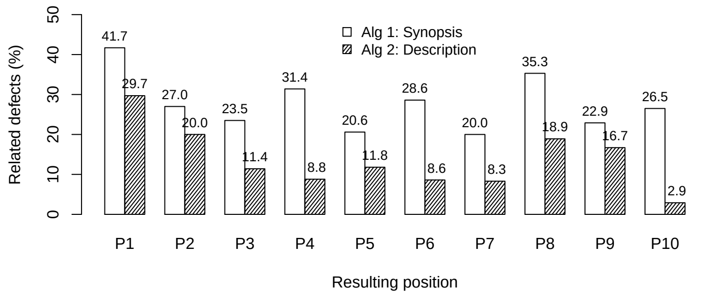


Figure 8.3: *Percentage of related defects per position.*

8.4 Attempt 4: Correlation between a related defect and its score per character

Another idea was to investigate the score–character ratio ($\frac{Similarityscore}{\#Characters} * 100$) and compare this ratio to the probability it is related. The motivation for this attempt was to determine whether high score–character ratio's

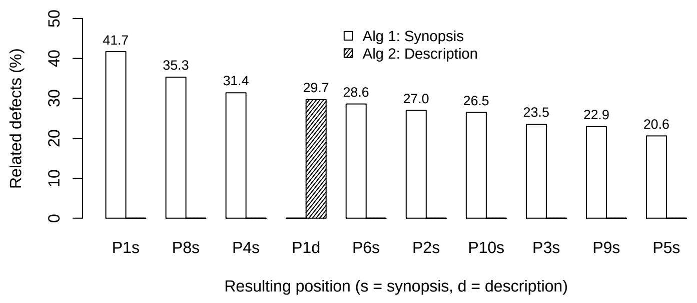


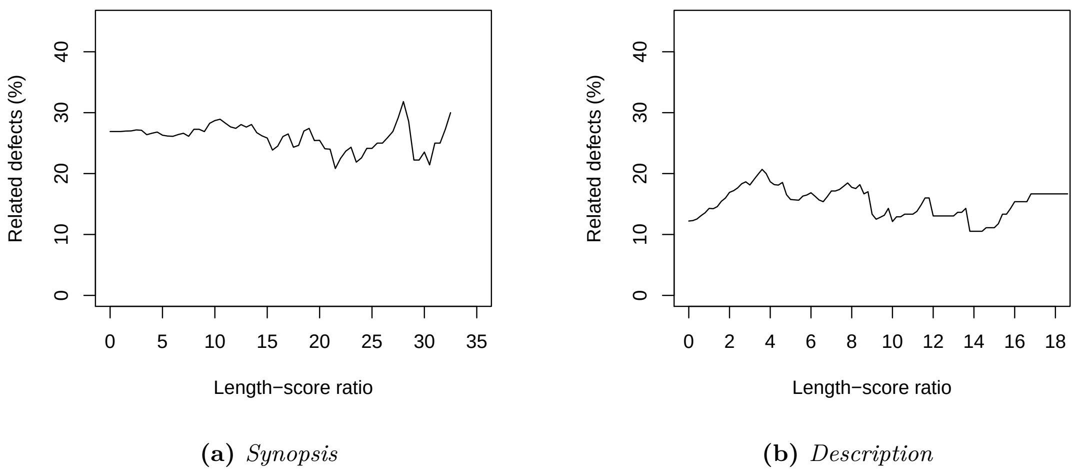
Figure 8.4: *Percentage of related defects per position (combined results)*

increase the probability that the defect is related. A high score–character ratio is achieved by having a relatively high idf score compared to the number of characters.

The resulting graph is shown in Figure 8.5. No clear pattern can be observed in the synopsis-based algorithm. We are searching for weaknesses of the synopsis-based algorithm. At a ratio of 22 and around a ratio of 30 the probability of the defect being related is relatively low. However, it seems to be a coincidence because both low ratio's are directly followed by a high peak. Therefore, we did not used this specific property to improve the algorithm. Furthermore, the lowest probability of finding a related defect using the synopsis-based algorithm is still higher than the highest probability of the description-based algorithm. As are result, no useful information was discover for combining the two algorithms.

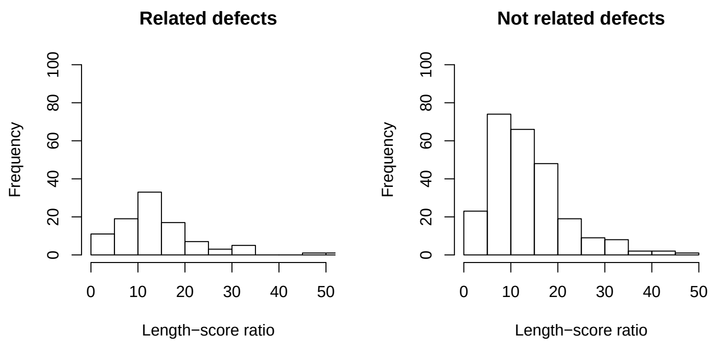
In Figure 8.6 the length–score ratio histograms are given. If we compare the two (related and not related) synopsis histograms to each other no clear difference between the distributions could be discovered. Also for the description-based algorithm no clear difference could be discovered.

Because these figures also did not provide any useful information we concluded that the length–score ratio was not a usable property for combining the two algorithms.

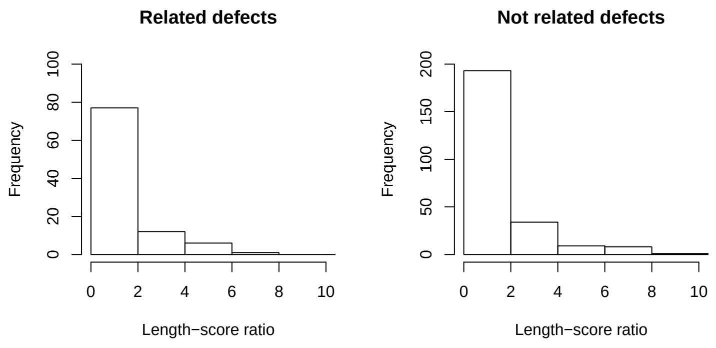


(a) *Synopsis* **(b)** *Description*

Figure 8.5: *Probability a defect is related according to the score–character ratio*



(a) *Synopsis*



(b) *Description*

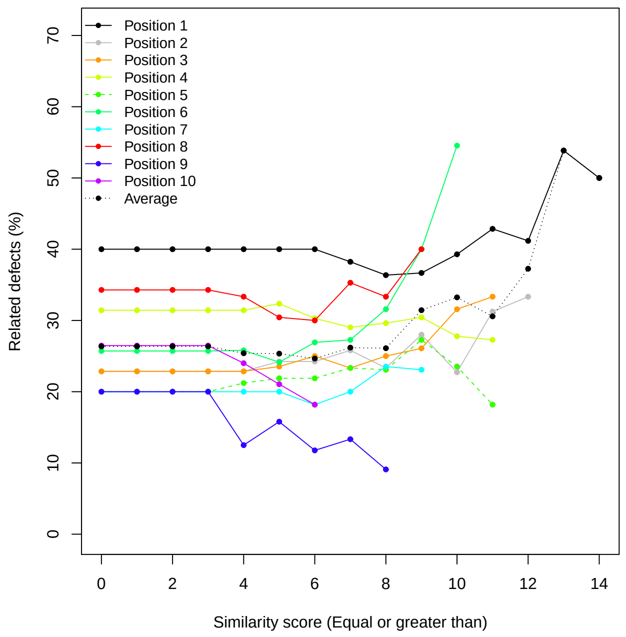
Figure 8.6: *Number of defect related/not related with respect to the score–character ratio*

8.5 Attempt 5: Correlation between a related defect and its score and position

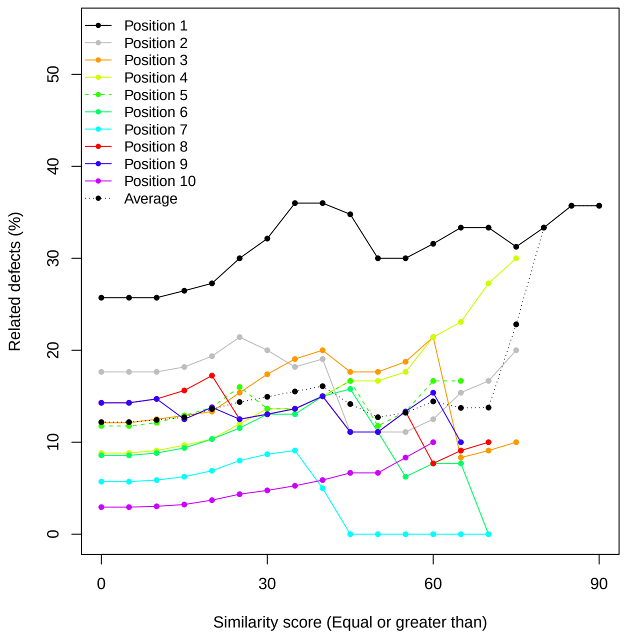
In this fifth attempt we investigate if a defect is related depending on its position and similarity score. In Figure 8.7a the probabilities for each (position, score) combination for the synopsis-based algorithm are shown. We observe that on average the probability that a defect is related becomes higher when the similarity score increases.

In Figure 8.7b the probabilities for each position, score combination for the description-based algorithm are shown. The first observation is that the number one result has by far the best probability compared to positions 2 to 10.

The average probability slightly increases till a similarity score of 40. The probability that a defect is related increases again when the score passes a value of 70. This strong increase is due to the fact that only four positions are part of the average value above a score of 70. For each of the other positions less than 10 results had a score above 70 which explains why we did not encountered these scores.



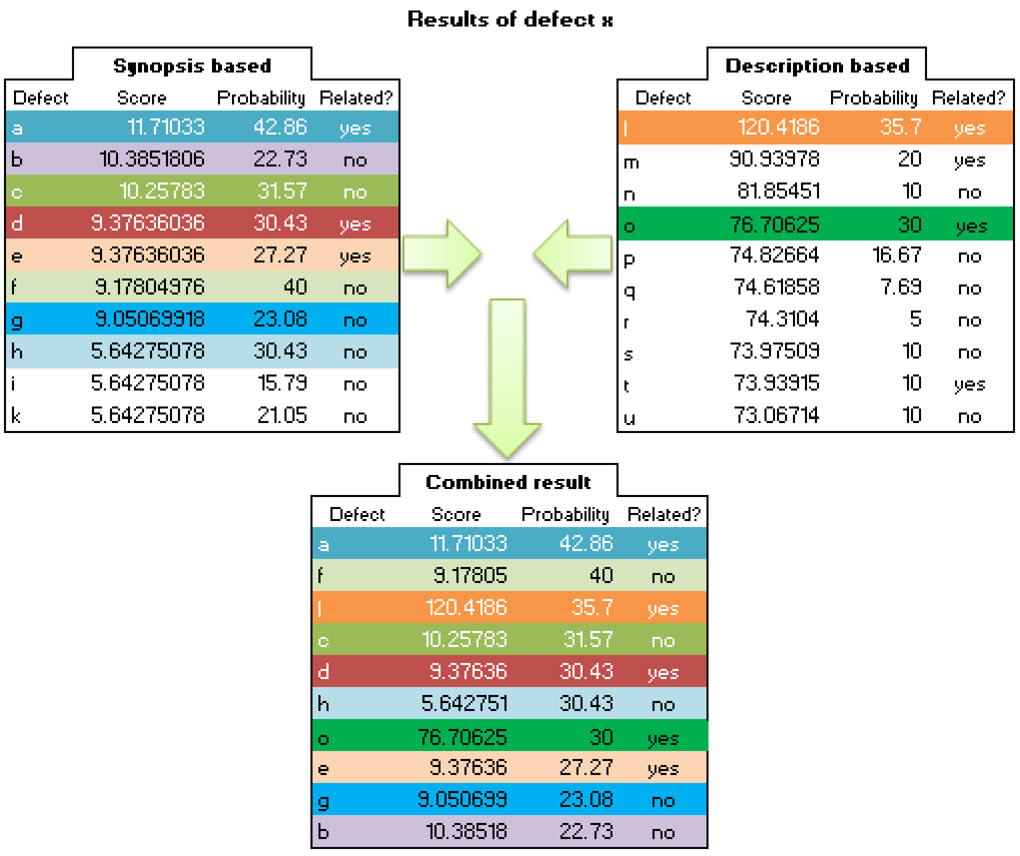
(a) *Synopsis*



(b) *Description*

Figure 8.7: *Probability a defect is related according its score and position*

Using these two figures (Figure 8.7a and 8.7b) we can match each individual result of the synopsis-based algorithm and the description-based algorithm to a probability it is related. The idea is to put the results of both algorithms in one result list and sort it on the probability in descending order. An example is given in Figure 8.8.



Results of defect x

Synopsis based

Defect	Score	Probability	Related?
a	11.71033	42.86	yes
b	10.3851806	22.73	no
c	10.25783	31.57	no
d	9.37636036	30.43	yes
e	9.37636036	27.27	yes
f	9.17804976	40	no
g	9.05069918	23.08	no
h	5.64275078	30.43	no
i	5.64275078	15.79	no
k	5.64275078	21.05	no

Description based

Defect	Score	Probability	Related?
l	120.4186	35.7	yes
m	90.93978	20	yes
n	81.85451	10	no
o	76.70625	30	yes
p	74.82664	16.67	no
q	74.61858	7.69	no
r	74.3104	5	no
s	73.97509	10	no
t	73.93915	10	yes
u	73.06714	10	no

Combined result

Defect	Score	Probability	Related?
a	11.71033	42.86	yes
f	9.17805	40	no
l	120.4186	35.7	yes
c	10.25783	31.57	no
d	9.37636	30.43	yes
h	5.642751	30.43	no
o	76.70625	30	yes
e	9.37636	27.27	yes
g	9.050699	23.08	no
b	10.38518	22.73	no

Figure 8.8: *Suggestion 5: Example of combining two result lists into one, using the probability it is related. The synopsis-based algorithm has a precision of 30% (3 defects related out of 10), the description-based algorithm has a precision of 40%. The precision after combining the results has increased to 50%. The colors show from which position in which algorithm the defect was taken.*

Results

Using this modified algorithm 2.84 related defects showed up in the result list. This resulted in an improvement of two percent, which seems to be quite limited. However, the theoretically maximum performance by combining the two algorithms is 3.70 related defects out of 10. This theoretically maximum was computed by combining the result lists of the two algorithms for each master defect in most optimal case.

This would be an improvement of 10.6%, which means we improved the algorithm by 18.9% in terms of the theoretical maximum possible improvement.

8.6 Conclusion

Of all optimization suggestions, the last suggestion performed the best. We had higher expectations about the optimization techniques. However, if we take a look at the distribution of all related results, it is not weird that we did not came close to the theoretically maximum performance. No strict correlations between a related defect and its score, length and/or position could be observed. Therefore, in a lot of cases a related result of the synopsis-based algorithm was replaced by a related result of the description-based algorithm which does not influenced the performance. Nevertheless, we gain an improvement of two percent. Therefore, this optimized algorithm is also implemented in the subsequent Related Defect Finder tool.

9 Related Defect Finder (RDF)

9.1 Introduction

In this chapter we discuss the 'Related Defect Finder' (RDF) tool. This tool is developed as a result of this thesis and is used at Océ to improve the bug fixing process. By entering a defect id, this tool returns relatively fast 10 probably related defects. The results should support the bug fixer by solving the defect or could help by determining if the defect entered is a duplicate defect.

9.2 Algorithms

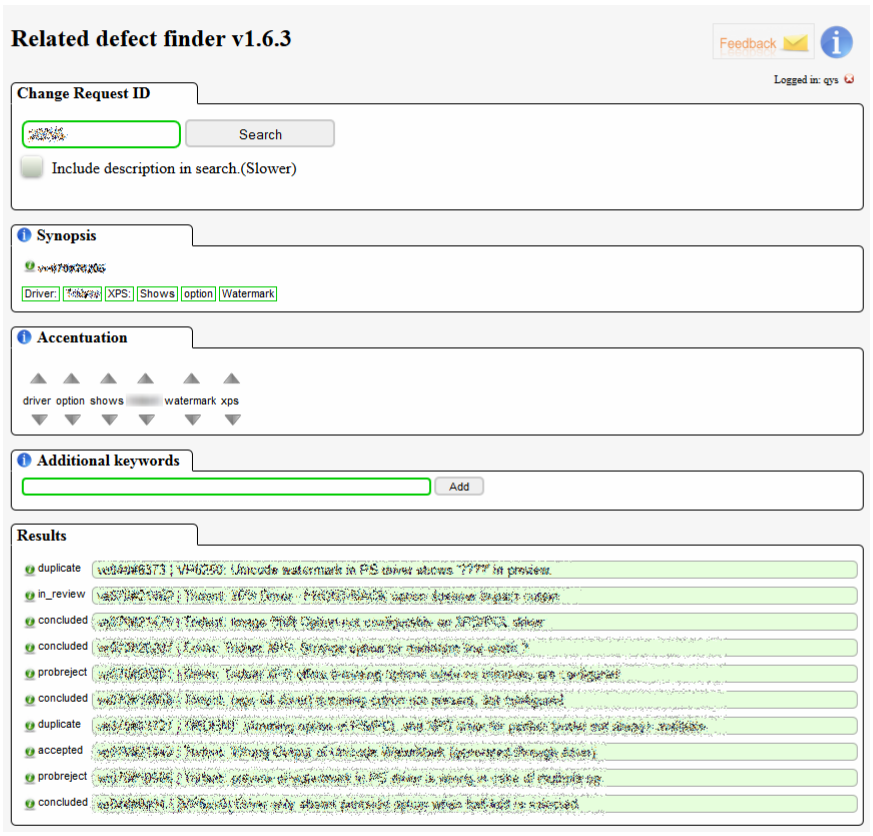
The results from the DRA tool showed that the synopsis-based algorithm performed best. Furthermore, the intersection of the synopsis-based algorithm results and description-based algorithm results was relatively small. Therefore, we decided to apply optimization techniques (Chapter 8) to improve the performance of the synopsis-based algorithm by combining it with the description-based algorithm. One of the optimizations proposals did actually improved the algorithm by 2%. In the RDF tool we implemented both algorithms. The motivation for implementing both algorithms was because the improvement of 2% takes ten times as much computation time compared to the synopsis-based algorithm. In Section 9.6 more details are given regarding the performance.

9.3 User interface

A screen shot of the RDF tool is shown in Figure 9.1. The screen is separated in five different parts. We discuss each of the individual parts below.

9.3.1 Change Request ID

In this part, the user can submit a defect id. Initially the result list of possibly related defects is computed by the synopsis based algorithm. However, in this first part, the user has the opportunity to check the 'Include description in search. (Slower)' checkbox. If checked, then the optimized algorithm, based on synopsis and description, is applied. This algorithm needs, as mentioned before, ten times as much time to compute compared to the synopsis-based algorithm.



Related defect finder v1.6.3

Feedback

Logged in: qys

Change Request ID

[illegible] Search

Include description in search.(Slower)

Synopsis

[illegible]

Driver: [illegible] XPS: Shows option Watermark

Accentuation

driver option shows [illegible] watermark xps

Additional keywords

Add

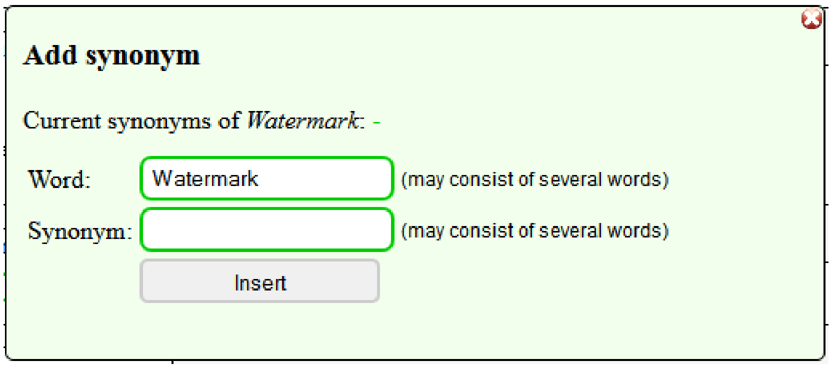
Results

duplicate	[illegible]
in_review	[illegible]
concluded	[illegible]
concluded	[illegible]
probreject	[illegible]
concluded	[illegible]
duplicate	[illegible]
accepted	[illegible]
probreject	[illegible]
concluded	[illegible]

Figure 9.1: *Screen shot of the RDF tool.*

9.3.2 Synopsis

This section shows the synopsis of the defect report which was requested in the first part. To distinguish between different terms, in this stage only a white space is used as delimiter. Terms are surrounded by a green border. No tokenization is applied because we would like to keep the synopsis in tact, which explains that in some cases symbols are present within the green border. By clicking a term, symbols are removed and the user can add a synonym to the current term (Figure 9.2). This added synonym is then also used for searching. Because the synonym is stored into a database it is used in each future search query. By clicking a term, already added synonyms to this term also become visible.



Add synonym

Current synonyms of *Watermark*: -

Word: Watermark (may consist of several words)

Synonym: (may consist of several words)

Insert

Figure 9.2: *Adding a synonym in the RDF tool.*

9.3.3 Accentuation

Less common words get according to the idf algorithm a relatively high idf score. However, in some specific situations, such as misspellings, it is preferable to decrease the idf score. The opposite also holds. Sometimes a common word could be very important in the search query, then it is preferable to increase the idf score.

In this part of the tool, one can influence the impact of each word. By clicking the arrow-down, the word is ignored for the search. By clicking the arrow-up the idf score is increased by 1, such that the emphasis of the word increases.

Increasing and decreasing the influence of a word can only be done once. By clicking a second time, the action is undone.

An example of influencing the weights can be seen in Figure 9.3.

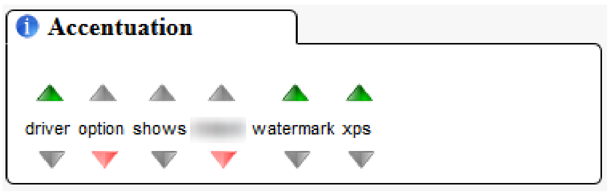


Figure 9.3: *Influencing weights in the RDF tool. When an arrow down is active(red) the word will be ignored for searching. When an arrow up is active(green) the idf score of the word will be increased by 1.*

9.3.4 Additional keywords

Sometimes the synopsis is very short (e.g. 'Error during shutdown'), which decreases the probability that a related defect is found,

For such cases, it could be beneficial to add additional keywords. Additional keywords are pre-processed in the same way as terms that are extracted from the synopsis / description.

Another way to use the additional keywords section is to add a word which may not appear in the result list. This approach can be used for instance if a product name shows up multiple times, while it cannot support solving the defect. By adding a minus ('-') before the word, the word is excluded. The similarity score of the result is in this case decreased by the idf value of the additional word.

9.3.5 Results

In this part of the tool the result list is displayed. For each of the results the current state (Figure 2.2), the synopsis and the defect id is given. The result list is updated each time a change is made in one of the other sections.

9.4 Logging

A lot of different aspects in the tool are interesting for future improvements. For example the accentuation part. If a word is marked by a number of different users and a certain number of times as not being relevant, the word could be, for instance, added to the stop word list. To ensure that such an improvement can be made in the future, the following events are logged:

- Search request (user name, datetime, defect-id, algorithm)
- Accentuations (user name, defect-id, term, influence, date-time)
- Additional keywords (user name, defect-id, term, is-Positive, date-time)

- Clicked results (user name, defect-id, clicked-defect-id, open-time, times-clicked, closed-time, position-result)
- Speed performance (date-time, performance, defect-id)

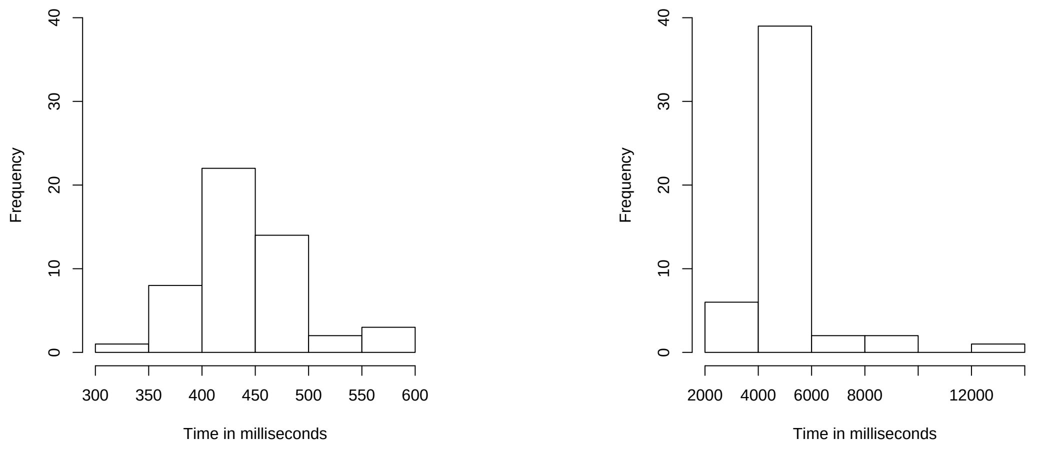
9.5 Synchronisation

In contrast to the DRA tool, the RDF tool has to be up-to-date at all time. At first all new entered defects have to be copied from the Synergy database to the RDF tool database. To prevent overloading the Synergy database we decided to periodically check for new defect reports. The RDF tool connects to Synergy to check for new added defects every 10 minutes. If a new defect has been found, all pre-processing steps (stemming, stop word filtering etc.) are executed and the defect is added to the database.

After a new defect is added, all idf values have to be updated. Indeed, the number of documents, which is part of the idf calculation, has changed. Therefore, the idf scores are also updated every 10 minutes. This action is completely integrated in the database, and takes less than a minute. Another property which has to be updated is the defect state. All defects from the last four weeks will be updated every minute. Because updating the statuses of the defects over the last two years is a quite heavy operation, this is done only once a day.

9.6 Performance

For the RDF tool we introduced a requirement which stated that calculation of the algorithm should take at most 10 seconds.



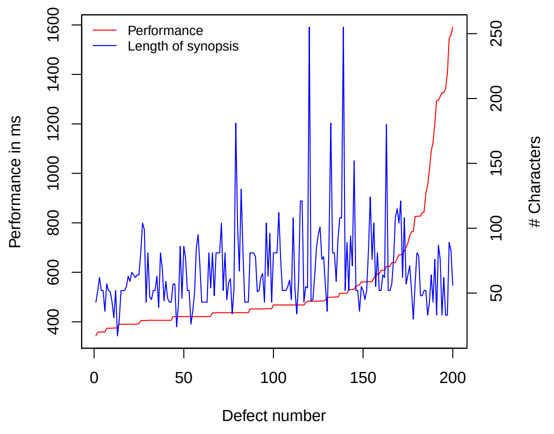
(a) *Synopsis-based algorithm* **(b)** *Optimized algorithm*

Figure 9.4: *Performance of computing the result list for the RDF tool*

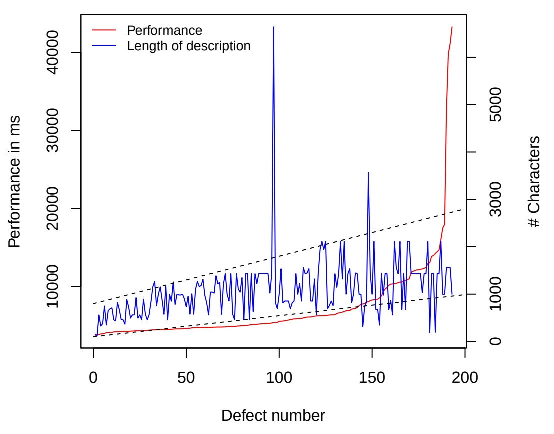
To determine if the algorithms are computed within 10 seconds, we tested the tool. We let our tool compute the result list for 50 different defect id's. We tested both algorithms using this approach. The results are shown in Figure 9.4. As we can see the optimized algorithm is approximately 10 times slower than the synopsis-based algorithm, which takes most of the time less than half a second. The optimized algorithm needs on average between four and six seconds, which is far below the limit of 10 seconds.

We expected to find a clear link between the performance and the length of the synopsis. However, after analyzing the result no clear relation was observed (Figure 9.5a). For the optimized algorithm we analyzed the relation between the performance and the length of the concatenation of synopsis and description. In

Figure 9.5b, a slightly increasing trend can be observed. However, the big outliers (defect number 99) did not influenced the graph at all. We also investigated the reason of the big outliers in performance. These outliers can be explained by a third party application which runs on the same server. Sometimes this third party application caused a 100% cpu usage which has consequences for the performance of the RDF tool.



(a) *Correlation between length of synopsis and performance of synopsis-based algorithm.*



(b) *Correlation between length of description and performance of the optimized algorithm.*

Figure 9.5: *Correlation between length of synopsis/description and performance of the optimized algorithm.*

9.7 Results

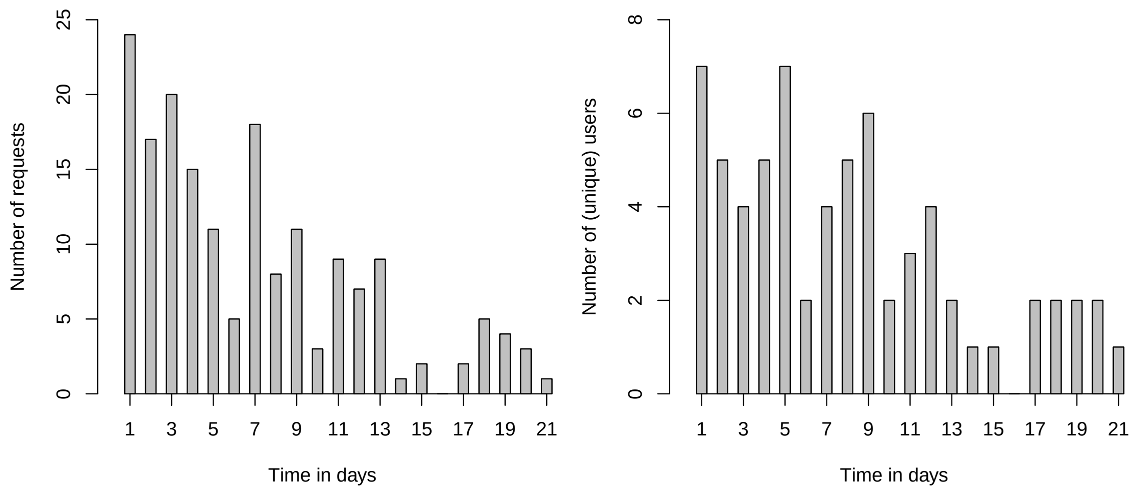
In this section we discuss the results of the RDF tool. We evaluated the results after an operational time of one month by observing the log data and conducting interviews. During this operational time of one month, the RDF tool was used by 29 different users. Figure 9.6 shows the usage per day of the RDF tool during the evaluation period. In Figures 9.6a and Figure 9.6b we can observe a decreasing trend. This is among other things due to the curiosity of persons for who the tool was not intended. Furthermore, not a lot of explicit actions were taken to promote the tool. An article was written for the intranet blog and a demonstration was given during a meeting with 10 team-members of different development teams. Unfortunately, it seemed that the blog was only consulted by a small number of people.

At the end of the evaluation period only two users still used the RDF tool. We interviewed these two users. The result of the interviews are discussed in Section 9.7.2.

Although one month of operational time is a short period, we do have some interesting results already. However, we cannot give any numbers of the performance quality. These can only be gathered after a longer period when actions in Synergy (state changes, relationship changes) can be related to a search request of the RDF tool. We did already found a number of RDF search requests which could be related to an action in Synergy. Two of them are discussed in Section 9.7.1. In Synergy, we also observed (during the evaluation period) a defect report was marked as duplicate five days after it was entered. If the RDF tool was consulted, the duplicate would have been found immediately because the master duplicate was present in the result list. This example confirms that the tool should be consulted more often than it is now.

9.7.1 Observation scenarios

We discuss two observations which show that our tool indeed supports the developers. These observations were established by comparing the RDF log results to the defect reports in IBM Rational Synergy. The developers



(a) *The number of defect search requests which were executed per day.* **(b)** *The number of unique users that accessed the RDF tool per day.*

Figure 9.6: *Usage of the RDF tool during the evaluation period of one month.*

mentioned in these observations are different from the experts which were involved in the voting process (Section 5.3). These developers started using RDF after it was promoted on the Océ intranet.

Observation 1: State change

In this first observation we were able to link a search request in RDF, to a state change in Synergy.

In this first step Developer 9 was searching for defects related to *ve070#20646* using RDF (Table 9.1). The optimized algorithm was selected to execute the search query. After submitting the search request, the result list in Figure 9.7 was shown to Developer 9.

Table 9.1: *Database entry of the defect request table.*

User name	Date time	Defect id	Algorithm
Developer 9	2013-08-07 18:44:44	ve070#20646	Optimized

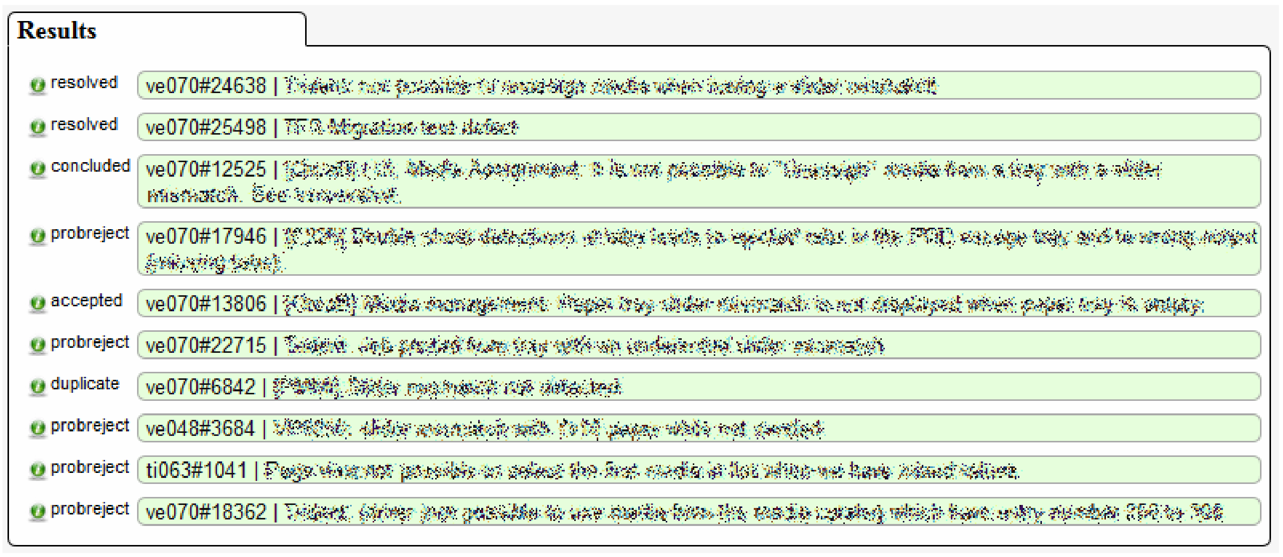
In Table 9.2 we can observe that Developer 9 opened defect id ve070#20638, which showed up at the first position, 15 seconds after submitting the search request.

Table 9.2: *Database entry of the clicked results table.*

User name	Requested defect id	Clicked defect id	Opening time	Closing time	Result position
Developer 9	ve070#20646	ve070#24638	2013-08-07 18:44:59	2013-08-07 18:46:50	1

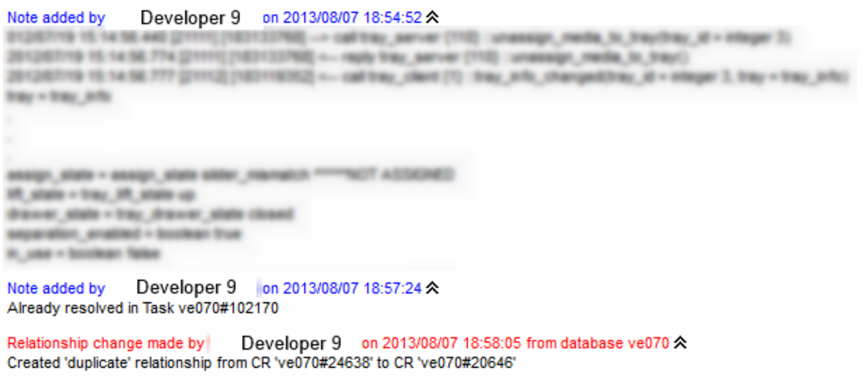
Finally we take a look at the transition log of the particular defect. In Figure 9.8 a fragment of the transition log is shown. In this transition log, the particular defect is marked as duplicate about 12 minutes after closing defect ve070#20638 by the same person, Developer 9.

Defect ve070#20646 was entered more than a year ago. Using the RDF tool Developer 9 observed that the defect was already fixed. The entire analysis took less than 14 minutes.


Results

resolved ve070#24638 | [redacted]
resolved ve070#25498 | [redacted]
concluded ve070#12525 | [redacted]
probreject ve070#17946 | [redacted]
accepted ve070#13806 | [redacted]
probreject ve070#22715 | [redacted]
duplicate ve070#6842 | [redacted]
probreject ve048#3684 | [redacted]
probreject ti063#1041 | [redacted]
probreject ve070#18362 | [redacted]

Figure 9.7: *Result list of ve070#20646 using the optimized algorithm.*


Note added by Developer 9 on 2013/08/07 18:54:52
[redacted]

Note added by Developer 9 on 2013/08/07 18:57:24
Already resolved in Task ve070#102170

Relationship change made by Developer 9 on 2013/08/07 18:58:05 from database ve070
Created 'duplicate' relationship from CR 've070#24638' to CR 've070#20646'

Figure 9.8: *Fragment of transition log of defect ve070#20646 in Synergy Rational Change. Task ve070#102170 belongs to defect ve070#24638.*

This observation shows that it is very likely RDF was used to solve the defect report. To validate the suspicion RDF was used, we interviewed Developer 9. During this interview Developer 9 confirmed our suspicion. The complete interview can be found in Appendix E.

Observation 2: Relationship added

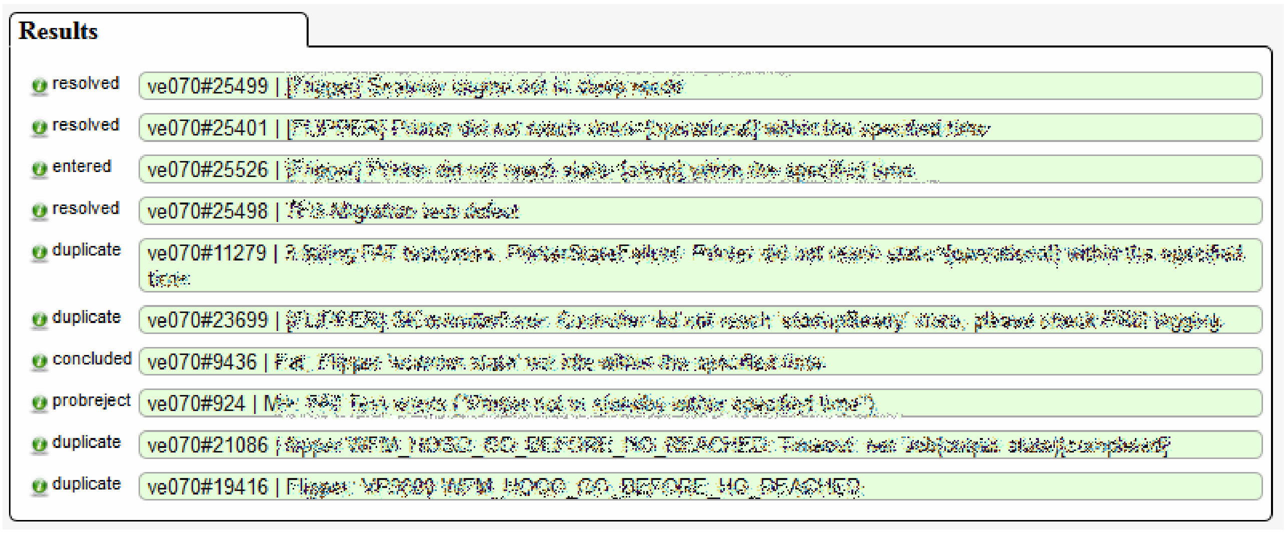
In this second observation we were able to link a search request to a relationship change in Synergy.

First, Developer 17 was searching for defects related to *ve070#25500* (Table 9.3). In this case the synopsis-based algorithm was selected. After submitting the search request, the result list in Figure 9.9 is shown to Developer 17.

Table 9.3: *Database entry of the defect request table.*

User name	Date time	Defect id	Algorithm
Developer 17	2013-08-26 09:36:37	ve070#25500	Synopsis-based

Table 9.4 shows that defect id 've070#25401' was opened three seconds after the results were shown by Developer 17. In this case, the second-best search result was opened.


Results

resolved	ve070#25499 | [illegible]
resolved	ve070#25401 | [illegible]
entered	ve070#25526 | [illegible]
resolved	ve070#25498 | [illegible]
duplicate	ve070#11279 | [illegible]
duplicate	ve070#23699 | [illegible]
concluded	ve070#9436 | [illegible]
probreject	ve070#924 | [illegible]
duplicate	ve070#21086 | [illegible]
duplicate	ve070#19416 | [illegible]

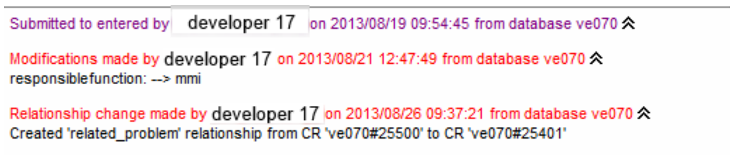
Figure 9.9: *Result list of ve070#25500 using the optimized algorithm.*

Figure 9.10 shows that approximately 30 seconds after opening the related defect, a relationship 'related problem' was added to defect report in IBM Synergy by Developer 17. Also in this scenario it was very likely that the RDF tool was used for the decision to add this relationship.

To confirm our suspicions we also interviewed Developer 17. Also in this interview our suspicions were confirmed by the developer. Developer 17 even mentioned that it should be possible to find a lot of other similar scenario's.

Table 9.4: *Database entry of the clicked results table.*

Username	Requested defect id	Clicked defect id	Opening time	Closing time	Result position
Developer 17	ve070#25500	ve070#25401	2013-08-26 09:36:49	2013-08-26 13:22:36	2


Submitted to entered by developer 17 on 2013/08/19 09:54:45 from database ve070

Modifications made by developer 17 on 2013/08/21 12:47:49 from database ve070
responsiblefunction: --> mmi

Relationship change made by developer 17 on 2013/08/26 09:37:21 from database ve070
Created 'related_problem' relationship from CR 've070#25500' to CR 've070#25401'

Figure 9.10: *Fragment of transition log of defect ve070#25500 in Synergy Rational Change.*

9.7.2 Interviews

The two scenarios discussed in Section 9.7.1 proved that RDF could be useful for developers. However, we do not know how often a useful defect report shows up in the result list and which algorithm is preferred. To answer these questions we interviewed the two most active users which are Developers 9 and 17 and two users who used the tool only a couple of times.

These four persons started using the tool after it was promoted on the blog, or were contacted by colleagues. We did not contacted them personally, therefore no extra knowledge was gathered by one of these four persons.

Developers 9 and 17 used the RDF tool for detecting related defects and duplicate defects. However, Developer 17 used the tool especially for detecting duplicate defects, while for Developer 9 RDF was used in most cases for detecting related defects.

The preferable algorithm was also different for both users. According to Developer 17, in most cases (according to Developer 17 approximately 90%), the result were the best if the optimized algorithm was used. The optimized algorithm performs the best when a stack trace or part of the system log is included in the description. On average, according to Developer 17, in about 7/10 searches a relevant result was found. Although the results of the optimized algorithm were only improved by 2%, the findings of Developer 17 confirm that combining synopsis and description is indeed very useful.

According to Developer 9, the synopsis-based algorithm performed the best. Only in some cases the description-based algorithm was used. On average at least one relevant result was found. Because of the positive performance results in terms of quality, Developer 9 recommended RDF to testers to reduce the number of duplicate defect reports.

The possibility to increase or decrease the influence of terms in the search algorithm has, according to Developer 9, a positive effect to the result list. Developer 17 also experienced that this feature has a positive effect to the results but he used this feature only sporadically.

Adding synonyms was only used by Developer 9, but only a couple of times. Two reasons were given for not using this feature. First, it was not quite clear how to add a synonym. Secondly, adding a synonym as an additional keyword is less time consuming while it has the same outcome.

According to both users, the ability to add extra keywords is a good extension to the tool. This extension offers the possibility to extend a short synopsis. The resulting log data can be used to match related words in the future, such that adding these extra keywords is, in a best case scenario, eventually not needed anymore. Furthermore, this extension is also used to check whether an observed defect is already submitted in Synergy by entering the synopsis into this field leaving the defect id field empty.

Both users were satisfied concerning the RDF performance in terms of speed. According to both users RDF is much faster than searching using Synergy. Furthermore, using this tool they both saved a lot of analyzing time.

In general more than 10 curious users tried the tool while it was not part of their work, such as managers and project leaders. Because defect fixing is not part of their job, it is not weird that the tool was used by them only sporadicly. We also interviewed two developers who used the tool only a couple of times. In these two interviews we focused on the reasons why these two developers stopped using RDF.

One of the developers was part of a team in which only a small number (less than 5) of defects is unresolved. RDF was used by the interviewee to test if it could be useful for his team. Because of the small number of defects, all unresolved defects are known by the team members. Therefore, RDF is not directly needed for this team.

The other interviewee used the tool once for a situation in which he knew a duplicate defect existed, but he was unable to find it. According to this interviewee the tool was fast and use friendly. However, it turned out that he was not aware of the synonym functionality. Furthermore, the accentuation part was a little bit unclear for him in the beginning. Therefore, we can conclude that some elements of the tool need some extra attention.

Because this interviewee was an expert on a particular subsystem, he was able to solve all other defects without any need of extra help which explains why the tool was not used more often. The interviewee also mentioned that using the tool more often could result in finding duplicate faster.

These two interviews (Appendix E) confirm that sporadicly use of the RDF tool does not imply that one is dissatisfied with the tool.

10 Conclusion

In this thesis we described an approach for detecting ‘related defects’. According to the definition in this thesis, a defect is related when the symptoms, the solution or the symptoms and solution are the same (duplicate). Because this relationship ‘is related’ was not stored in the defect repository, we had to construct an oracle. However, to construct an oracle we had to evaluate almost 800 million comparisons by hand. As this operation is infeasible to do by hand, we decided to take an alternative approach. In this alternative approach we first developed three different algorithms. Then we implemented these three algorithms in a tool called DRA. To prove that our developed algorithms perform better than a random selection we added a random selection to the results. Five experts were asked to evaluated the results of 10 different given defects using the DRA tool. The results showed that the interpretation of related differs between experts with a different number of years of experience. These differences in interpretation showed that the notion of ‘related’ differs per person.

The results also revealed that the synopsis-based algorithm achieved the best results in terms of quality. The intersection between the synopsis-based and description-based algorithm was less than 3%, therefore we decided to combine these algorithms. Another reason for combining these algorithms was the fact that in case of a crash often a log is copied into the description field. Logs contain specific information such as function names which increases the probability of a match.

The resulting two algorithms were implemented in a tool called RDF. This tool is available for all software developers within Océ R&D. After an evaluation period of one month, we evaluated the log results and interviewed four users.

During the evaluation period two users used the tool frequently from which we can conclude that RDF is indeed useful. However, most developers are accustomed to a particular working method. It is hard to change this particular way of working. Therefore, as an alternative the tool could be recommended to starters who not have a particular working method yet.

We can conclude that related defects can indeed be helpful in solving defects. However, it is hard to convince developers that the tool can be a useful supplement in the process of solving defects. Therefore, more developers have to be interviewed to get a better picture of their reasons for not using the tool.

Threats to validity

The results of the DRA tool were evaluated by only five experts with different levels of experience and differed a lot between experienced and less experienced experts. The definition of the notion of ‘related defect’ also varies per person. Moreover, only master defects of a particular subsystem were evaluated. Evaluating master defects of other subsystems could have affected the results.

In total a maximum of 40 results had to be evaluated by the experts for each of the master defects. As a consequence fatigue or loss of concentration may affect the outcome. Therefore, an error margin has to be taken into account for the results of the DRA tool.

The results of DRA were also used to determine the best performing algorithm and to optimize the best performing algorithm. The optimized algorithm performed only 2% better than the synopsis-based algorithm. For this small improvement we also have to consider a margin of error. Therefore, it is possible that the optimized algorithm does not perform better than the synopsis-based algorithm.

According to a frequent user of the RDF tool, for every search a related defect is found using the synopsis-based algorithm. According to another frequent user, a related defect is found in 70% of all searches using the optimized algorithm. These results show that both algorithms perform good in terms of quality. The interviewees who used the tool less frequent were not dissatisfied with the results. One of these interviewees mentioned that the tool was not interesting for his tasks. The other interviewee was an expert who did not needed the help of related defects. However, it could be that according to other less frequent users the performance of the tool is indeed below their expectations.

Future work

The evaluation period was just one month. Because the tool was not used as frequently as expected, not enough data was gathered to present some statistics regarding time savings or detected related defects. As future work, the RDF tool could be extended such that more software departments within Océ could use the tool. This should increase the number of active users and should provide more log data. Furthermore, a format / template for filling in synopsis and description could be introduced. This prevents synopses from being too short or could result in better matches for the description field (e.g. by always adding a log to the description field).

Because the system log is added many more times to defect reports over the last year, it could be used for searches in recent defects. To improve the results, the more detailed trace log and protocol log could also be used. These logs are very low level, therefore both logs have to be analyzed in detail.

In the near future, Océ will switch to a new software package instead of IBM Rational Synergy. Therefore, an extension for the RDF tool could be implemented such that it supports both, the new software package and IBM Rational Synergy.

Another improvement could be to extend the RDF tool such that log data is directly used to improve the results. This could be done for instance by automatically adding frequently negatively rated words to the stop word list. The opposite could also be an improvement. The idf-score of a term which is rated positively multiple times could be increased in idf-score automatically on a certain threshold. Currently, 282 unique words are rated. The most frequently rated word was rated only 8 times. In total, 381 ratings were issued within 50 days. A minimum number of ratings and a confidence factor should be determined to automatically process these ratings.

According to the interviewees, defect reports older than one month are in most cases not relevant. To optimize the result list, RDF could be extended by adding an option to enter a period which filters irrelevant older defects. Applying such a filter decreases the data set, which also improves the performance in terms of speed. One of the interviewees mentioned that only statuses ‘duplicate’, ‘resolved’ and ‘concluded’ are relevant. A solution for this statement is to add a status filter which filters irrelevant statuses from the result list.

The interviews also revealed that it is sometimes desirable to show more than 10 results. Especially in cases when it is difficult to add some relevant additional keywords. Thus, an option to show more than 10 result could also be added.

The ability for the addition of synonyms seemed not to be user-friendly as users preferred to use the additional keywords functionality. In order to ensure that that synonyms are added more often, the user interface could be redesigned.

Relationships (e.g. ‘is duplicate of’ or ‘is related to’) stored in Synergy are currently not used to compose the result list. As an improvement these relationships could also be taken into account. The main advantage is that these relationships are already verified.

Bibliography

[AKAD$^+$13] Mehdi Amoui, Nilam Kaushik, Abraham Al-Dabbagh, Ladan Tahvildari, Shimin Li, and Weining Liu. Search-based duplicate defect detection: an industrial experience. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 173–182, Piscataway, NJ, USA, 2013. IEEE Press.

[BCA12] Sean Banerjee, Bojan Cukic, and Donald Adjeroh. Automated duplicate bug report classification using subsequence matching. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 74–81, 2012.

[Bra09] Ígor A. Braga. Evaluation of stopwords removal on the statistical approach for automatic term extraction. In *Information and Human Language Technology (STIL), 2009 Seventh Brazilian Symposium in*, pages 142–149, 2009.

[HGPVS13] Matthew J. Howard, Samir Gupta, Lori Pollock, and K. Vijay-Shanker. Automatically mining software-based, semantically-similar words from comment-code mappings. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 377–386, Piscataway, NJ, USA, 2013. IEEE Press.

[JPPC12] Hardik Joshi, Jyoti Pareek, Rushikesh Patel, and Krunal Chauhan. To stop or not to stop x2014 - experiments on stopword elimination for information retrieval of gujarati text documents. In *Engineering (NUiCONE), 2012 Nirma University International Conference on*, pages 1–4, 2012.

[JW08] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61, 2008.

[KMM00] Mark Kantrowitz, Behrang Mohit, and Vibhu O. Mittal. Stemming and its effects on tfidf ranking. In *SIGIR*, pages 357–359, 2000.

[LM13] Johannes Lerch and Mira Mezini. Finding duplicates of your yet unwritten bug report. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 69–78, 2013.

[LT11] Ying Liu and Zheng Tie. Automatic extraction and filtration of multiword units1. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 4, pages 2591–2595, 2011.

[MBK91] Yoëlle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. An information retrieval approach for automatically constructing software libraries. *Software Engineering, IEEE Transactions on*, 17(8):800–813, 1991.

[MGL09] Prem Melville, Wojciech Gryc, and Richard D. Lawrence. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 1275–1284, New York, NY, USA, 2009. ACM.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*, pages 232–234. Cambridge University Press, New York, NY, USA, 2008.

[MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, 1999.

[NC08] Hassiba Nemmour and Youcef Chibani. New Jaccard-distance based support vector machine kernel for handwritten digit recognition. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–4, 2008.

[NNN+12] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 70–79, 2012.

[Pai90] Chris D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, 1990.

[Por80] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Por01] Martin Porter. Snowball: A language for stemming algorithms. http://snowball.tartarus.org/texts/introduction.html, October 2001. Accessed 02.04.2013.

[Ran] RanksNL. English stopwords list. http://www.ranks.nl/resources/stopwords.html. Accessed: 26.04.2013.

[RAN07] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 499–510, 2007.

[RRGACD12] Thomas J. Ramirez-Rozo, Julio C. Garcia-Alvarez, and Germán Castellanos-Dominguez. Infrared thermal image segmentation using expectation-maximization-based clustering. In *Image, Signal Processing, and Artificial Vision (STSIVA), 2012 XVII Symposium of*, pages 223–226, 2012.

[RZT04] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 42–49, New York, NY, USA, 2004. ACM.

[SAMO11] Siti Salwa Salleh, Noor Aznimah Abdul Aziz, Daud Mohamad, and Megawati Omar. Combining Mahalanobis and Jaccard to improve shape similarity measurement in sketch recognition. In *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on*, pages 319–324, 2011.

[SJ10] Ashish Sureka and Pankaj Jalote. Detecting duplicate bug report using character n-gram-based features. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 366–374, 2010.

[SLKJ11] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 253–262, 2011.

[SLW+10] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 1, pages 45–54, 2010.

[Tec13] Océ Technologies. Defect working method, 2013.

[Tim12] New York Times. Knight capital says trading mishap cost it 440 million. http://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/, 2012. Accessed: 24.06.2013.

[vDBG04] Sebastian van Delden, David B. Bracewell, and Fernando Gomez. Supervised and unsupervised automatic spelling correction algorithms. In Du Zhang, Éric Grégoire, and Doug DeGroot, editors, *IRI*, pages 530–535. IEEE Systems, Man, and Cybernetics Society, 2004.

[WZX$^+$08] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, pages 461–470, 2008.

[ZCT$^+$04] Hugo Zaragoza, Nick Craswell, Michael J. Taylor, Suchi Saria, and Stephen E. Robertson. Microsoft cambridge at trec 13: Web and hard tracks. In Ellen M. Voorhees and Lori P. Buckland, editors, *TREC*, volume Special Publication 500-261. National Institute of Standards and Technology (NIST), 2004.

A
Appendix: Defect Relationship Allocator(DRA) User manual

This user manual was provided to the experts who had to evaluate the results of the RDF tool (Chapter 7).

USER MANUAL *"DEFECT RELATIONSHIP ALLOCATOR"*

PROJECT PURPOSE

Fixing defects is quite time consuming, especially for beginners. At Océ, defects over the last decade are all stored in IBM Rational Synergy. This repository contains a lot of useful information but has never been used to improve the bug fixing process. For my master thesis I will use this data to develop an algorithm which automatically searches for 'related' defect reports. The related defects should help the bug fixer by solving a defect. The dataset however does not contain information about related defects. Therefore we need a tool to accomplish this.

PURPOSE OF THE TOOL

The "Defect Relationship Allocator" is developed to assign relationships between defects. The results will be used to determine which of the developed algorithms performs the best in finding related defects. I do not want to influence the result in any possible way, therefore details about the algorithms will not be given in this manual.

BRIEF EXPLANATION

Each time you refresh the website (or press) a new defect will be shown on top of the page. We will call this defect the `master defect`. We ask you to Like ( ) or Dislike ( ) each of the resulting defects which are shown in the list in yellow.

WHEN DO I PRESS THE LIKE ( ) BUTTON?

If the defect on the same row as the -button is related to the master defect, click on the -button.

WHEN DO I PRESS THE DISLIKE ( ) BUTTON?

If the defect on the same row as the -button is **not** related to the master defect, click on the -button.

WHEN IS A DEFECT RELATED?

A defect is related to the master defect if the defect would help the bug fixer to solve the master defect.

For example:

- It is a duplicate of the master defect
- The defects describe the same problem but the products on which the bug occurs differ.
- The defects have similar solutions (e.g. both are deadlocks that occur due to releasing interfaces in the wrong order).
- The defects have similar symptoms (e.g. both defects result in incorrect contradictions in mixed orientation jobs)

If you are in doubt, imagine that you are assigned to resolve the master defect: which of the listed defects help you to accomplish this?

I CAN ONLY SEE THE SYNOPSIS, IS THERE A POSSIBILITY TO ALSO SEE THE DESCRIPTION?

Yes, there is. Each defect (including the master) has a ✓-icon. When pressing this icon (or click on the synopsis) the icon becomes green and the defect is shown in synergy-layout. If you are not yet logged in you have to enter your synergy credentials once.

GETTING STARTED

1. Browse to http://ws050755:8082/dra/ (Internet Explorer, Google Chrome) (Firefox not supported)
2. If this is the first time you visit the website, then you have to enter you Océ username and the number of months of experience you have with the wfm subsystem. When finished, press submit.
3. You are logged in now. In the right-hand corner, you can find your username and months of experience. If this information is incorrect, then you can press the ✖-button. You can then reenter you username and experience.
4. In the box on the top the master defect will be loaded, this may take a few seconds. After loading the master defect, the defect list will be loaded. This also can take some time.
5. On top of the loaded list the number of finished defects is shown. You will get an extra message if you finished 10 master defects.
6. You can now start voting. Votes are saved immediately to the database.
7. When you have entered a vote for each of the defects you can press *"F5"* or press the ↻-button. A new master defect and defect list will then be loaded. The number of finished defects will be incremented by 1.

B

Appendix: List of misspellings

Table B.1: *Commonly misspelled words with their corresponding correction.*

absense	absence	amatuer	amateur	capital	capitol
absance	absence	amature	amateur	carrer	career
acceptible	acceptable	ammend	amend	carefull 5	careful
accidentaly	accidently	anually	annually	Carribean	Caribbean
accidentally	accidently	annualy	annually	catagory	category
accomodate	accommodate	apparant	apparent	cauhgt	caught
acommodate	accommodate	aparent	apparent	caugt	caught
acheive	achieve	artic	arctic	cemetary	cemetery
acknowlege	acknowledge	arguement	argument	cematery	cemetery
aknowledge	acknowledge	athiest	atheist	changable	changeable
acquaintence	acquaintance	avrage	average	cheif	chief
aquaintance	acquaintance	averege	average	collaegue	colleague
aquire	acquire	awfull	awful	collegue	colleague
adquire	acquire	aweful	awful	collectable	collectible
aquit	acquit	ballance	balance	colum	column
acrage	acreage	balence	balance	confortable	comfortable
acerage	acreage	basicly	basically	comming	coming
adress	address	becuase	because	commited	committed
adultary	adultery	becomeing	becoming	comitted	committed
adviseable	advisable	begining	beginning	compitition	competition
advizable	advisable	beleive	believe	conceed	concede
agression	aggression	bellweather	bellwether	concieve	conceive
aggresion	aggression	bouy	buoy	congradulate	congratulate
alchohol	alcohol	bouyant	buoyant	consciencious	conscientious
algorythm	algorithm	burgler	burglar	concious	conscious
alege	allege	bisness	business	consious	conscious
allage	allege	bussiness	business	concensus	consensus
allegaince	allegiance	bizness	business	contraversy	controversy
allegience	allegiance	buisness	business	conveniance	convenience
alegiance	allegiance	calender	calendar	cooly	coolly
allmost	almost	camoflage	camouflage	critecize	criticize
alot	lot	camoflague	camouflage	critisize	criticize

dacquiri	daiquiri	hypocrisy	hypocrite	neice	niece
daquiri	daiquiri	ignorence	ignorance	nieghbor	neighbor
decieve	deceive	imaginery	imaginary	noticable	noticeable
definate	definite	immitate	imitate	occassion	occasion

definit	definite	immitation	imitation	occasionaly	occasionally
definitly	definitely	imitashun	imitation	occassionally	occasionally
definately	definitely	imediately	immediately	occurrance	occurrence
defiantly	definitely	incidently	incidentally	occurence	occurrence
desparate	desperate	independant	independent	occured	occurred
diffrence	difference	indispensible	indispensable	ommision	omission
dilema	dilemma	innoculate	inoculate	omision	omission
disapoint	disappoint	inteligence	intelligence	orignal	original
disasterous	disastrous	intelligance	intelligence	outragous	outrageous
drunkeness	drunkenness	intresting	interesting	parliment	parliament
dumbell	dumbbell	interuption	interruption	passtime	pastime
embarass	embarrass	irrelevent	irrelevant	pasttime	pastime
equiptment	equipment	irritible	irritable	percieve	perceive
excede	exceed	iland	island	perseverence	perseverance
exilerate	exhilarate	it's	its	personell	personnel
existance	existence	jellous	jealous	personel	personnel
experiance	experience	jewelery	jewelry	plagerize	plagiarize
extreem	extreme	jewellery	jewelry	playright	playwright
facinating	fascinating	judgement	judgment	playwrite	playwright
firey	fiery	kernal	kernel	posession	possession
flourescent	fluorescent	liesure	leisure	possesion	possession
foriegn	foreign	liason	liaison	potatos	potatoes
freind	friend	libary,liberry	library	preceed	precede
fullfil	fulfil	lisence	license	presance	presence
fulfill	fulfil	licence	license	principal	principle
guage	gauge	lightening	lightning	privelege	privilege
gratefull	grateful	maintenence	maintenance	priviledge	privilege
greatful	grateful	maintainance	maintenance	professer	professor
garantee	guarantee	maintnance	maintenance	promiss	promise
garentee	guarantee	medeval	medieval	pronounciation	pronunciation
garanty	guarantee	medevil	medieval	prufe	proof
guidence	guidance	mideval	medieval	prophesy	prophecy
harrass	harass	momento	memento	publically	publicly
heighth	height	millenium	millennium	quarentine	quarantine
heigth	height	milennium	millennium	que	queue
heirarchy	hierarchy	miniture	miniature	questionaire	questionnaire
horsderves	horsd'oeuvres	miniscule	minuscule	questionnair	questionnaire
ordeurves	horsd'oeuvres	mischievious	mischievous	readible	readable
humerous	humorous	mischevous	mischievous	realy	really
hygene	hygiene	mischevious	mischievous	recieve	receive
hygine	hygiene	mispell	misspell	reciept	receipt
hiygeine	hygiene	misspel	misspell	recomend	recommend
higeine	hygiene	neccessary	necessary	reccommend	recommend
hipocrit	hypocrite	necessery	necessary	refered	referred
referance	reference	wellfare	welfare		
refrence	reference	welfair	welfare		
relevent	relevant	wether	whether		
revelant	relevant	wilfull	wilful		
religous	religious	willful	wilful		
religius	religious	withold	withhold		
repitition	repetition	writting	writing		
restarant	restaurant	writeing	writing		
restaraunt	restaurant	you're	your		
rime	rhyme				
ryme	rhyme				

rythm	rhythm
rythem	rhythm
secratary	secretary
secretery	secretary
sieze	seize
seperate	separate
sargent	sergeant
similer	similar
skilfull	skilful
skillful	skilful
speach	speech
speeche	speech
succesful	successful
successfull	successful
sucessful	successful
supercede	supersede
suprise	surprise
surprize	surprise
they're	they
tomatos	tomatoes
tommorrow	tomorrow
twelth	twelfth
tyrany	tyranny
underate	underrate
untill	until
upholstry	upholstery
usible	usable
useable	usable
useing	using
usualy	usually
vaccuum	vacuum
vacume	vacuum
vegatarian	vegetarian
vehical	vehicle
visious	vicious
wether	weather
wierd	weird

C SQL queries to compute the related defect result list.

Listing C.1: *SQL Query to compute related defects using EXISTS. Computation time: 12.6 seconds on average*

```
SELECT a.crid_full, synopsis as combi, synopsis,a.s
FROM
        (
         SELECT isyn.crid_full, sum(isyndes.score) as s
         FROM idfscore isyn INNER JOIN idfscore_syndes isyndes ON
                isyn.crid_full = isyndes.crid_full AND
                isyn.term = isyndes.term
         WHERE isyn.crid_full != 've070#17161' AND
                 EXISTS
                (
                 SELECT b.term
                 FROM idfscore b
                 WHERE b.crid_full = 've070#17161' AND
                 b.term = isyn.term
                )
         GROUP BY crid_full
         ORDER BY sum(isyndes.score) DESC
         LIMIT 10
        ) a
    JOIN change_request ON change_request.crid_full = a.crid_full
\label{lst:Appendix:sqlqueriesExists}
```

Listing C.2: *SQL Query to compute related defects using inner join. Computation time: 0.2 seconds on average*

```
SELECT
        a.crid_full, concat(synopsis, ' ', description) as combi, synopsis, a.s
FROM
        (
         SELECT a.crid_full, sum(c.score) as s
         FROM idfscore a
         INNER JOIN
                (
                 SELECT b.term, b.score
                 FROM idfscore_syndes b
                 INNER JOIN
                        (
                         SELECT e.term, e.score
                         FROM idfscore e
                         WHERE e.crid_full = 've070#17161'
                        ) as d
                 ON d.term = b.term
                 WHERE b.crid_full = 've070#17161'
                ) as c
         ON c.term = a.term
         WHERE a.crid_full != 've070#17161'
         GROUP BY crid_full
         ORDER BY sum(c.score) DESC
         LIMIT 10
        ) a
JOIN change_request ON change_request.crid_full = a.crid_full
\label{lst:Appendix:sqlqueriesInnerJoin}
```

D

Appendix: Stop words list

Table D.1: *Stopword list [Ran] which is used in DRA and RDF extended with template words. Template words which were not included in the Ranks.nl stop word list are formatted bold.*

a	**cep**	here	most	she'll	through	with
about	could	here's	mustn't	she's	to	won't
above	couldn't	hers	my	should	too	would
actual	did	herself	myself	shouldn't	under	wouldn't
after	didn't	he's	no	so	until	you
again	do	him	nor	some	up	you'd
against	does	himself	not	**specified**	**version**	you'll
all	doesn't	his	of	**state**	very	your
am	doing	how	off	**steps**	was	you're
an	don't	how's	on	such	wasn't	yours
and	down	i	once	**system**	we	yourself
any	during	i'd	only	than	we'd	yourselves
application	each	if	or	that	we'll	you've
are	**engine**	i'll	other	that's	were	
aren't	**expected**	i'm	ought	the	we're	
as	few	in	our	their	weren't	
at	for	into	ours	theirs	we've	
be	from	is	ourselves	them	what	
because	further	isn't	out	themselves	what's	
been	had	it	over	then	when	
before	hadn't	its	own	there	when's	
behaviour	has	it's	**preconditions**	there's	where	
being	hasn't	itself	*******	these	where's	
below	have	i've	**productconfig**	they	which	
between	haven't	let's	**reduce**	they'd	while	
both	having	**logging**	**relevant**	they'll	who	
but	he	**logserver**	same	they're	whom	
by	he'd	**manuallogs**	shan't	they've	who's	
cannot	he'll	me	she	this	why	
can't	her	more	she'd	those	why's	

E Appendix: Interviews

This appendix contains the elaboration of the face-to-face interviews which were held after the one month of evaluation of the RDF tool.

Interview RDF tool Developer 9

1. **Wat was je eerste indruk van de tool? Was alles meteen duidelijk, of was het nodig om de handleiding te lezen?**
   What was your first impression of the tool? Was everything immediately clear or was it necessary to read the manual?

   Developer 9**:** *In the beginning I did not noticed that a checkbox was present to select the description based search algorithm. Overall the tool was clear.*

2. **Gebruik je de tool voor het zoeken van gerelateerde defecten, of voor andere dingen zoals bijvoorbeeld het zoeken naar duplicates?**
   Do you use the tool for detecting related defects, or for some other reasons such as finding duplicates?
   Developer 9: *I use the tool in most cases for detecting related defects. Sometimes I also uses it for finding duplicates.*

3. **Welke zoekfunctie (met of zonder description) levert het beste resultaat op?**
   Which search algorithm (with or without description) delivers the best results?

   Developer 9: *I use the default algorithm the most, which is the algorithm without description. It is very fast and I think the results are better than the extended algorithm.*

4. **Wat vind je van de resultaten die de tool geeft? Zit er altijd een zinvol resultaat bij, of is dit slechts sporadisch het geval? Nu worden er tien resultaten getoond, is dit voldoende?**
   What do you think of the results indicated by the tool? Is there always a meaningful result included, or is this only occassionaly the case? Ten results shown, is this sufficient?

   Developer 9: *The results are good. At least one related defect is found in each result list. I also recommended the tool to testers to check for duplicates before entering a defect. In this case the additional keywords textbox is useful.*

5. **In de lijst van resultaten is ook de huidige status van het CR gegeven. Gebruik je deze informatie, of is deze niet relevant? Zou het bijvoorbeeld handig kunnen zijn om de lijsten per status te grouperen.**
   In the list of results also the current status of the CR is given. Do you use this information, or is it irrelevant? Could it be useful to group the result lists by their status?

   Developer 9: *It could be useful, but entering a timespan would be more useful. Currently very old defects are also detected, which are most of the time not useful.*

6. **Wat vind je van de performance, werkt de tool snel genoeg?**
   How do you think about the performance, is the tool fast enough?

   Developer 9: *It works very fast. Much faster than a search using Synergy.*

7. **Heb je de functionaliteit om synonymen toe te voegen gebruikt?**
   Did you used the functionality to add synonyms?

   Developer 9: *I used it a couple of times, but most of the times I used the additional keywords to add some other words, because it is faster. Furthermore, the functionality is not easy to find.*

8. **Heb je de functionaliteit om accenten toe te voegen wel eens gebruikt? Zo ja, werden de resultaten hierdoor zinvoller?**
   Did you used the functionality to add accents? If so, did the results became more useful?

   Developer 9: *Yes, I used it a lot. Especially in cases of a long synopsis, decreasing the influence of words is very useful. Increasing and decreasing the influence are used a same number of times.*

9. **Heb je de functionaliteit om additionele zoekworden toe te voegen wel eens gebruikt? Zo ja, werden de resulaten hierdoor zinvoller?**
   Did you used the functionality to add additional keywords? If so, did the results became more useful?

   Developer 9: *Yes, for example when a synopsis is quite short or for adding some synonyms. I also used this functionality before entering a new defect report to prevent from adding duplicate defect reports. That is why I recommended this tool to the testers.*

   Sometimes, this textbox is used to find a solution for a problem which is already solved in another build.

10. **Heb je ook de functionaliteit gebruikt om negatieve zoekwoorden toe te voegen? Was je van deze functionaliteit op de hoogte? Werden de resulaten hierdoor zinvoller?**
    Did you used the functionality to add negative keywords? Were you aware of this functionality? Did the results became more useful?

    Developer 9: *I did not notice this functionality. However, it could be useful if you are searching for a problem on a particular product.*

11. **Wat zou je willen verbeteren aan de tool?**
    Do you have suggestions to improve the tool?

    Developer 9:
    - *Also add already marked related defects to the result list. (If searching for A and A is related to B, then add B to the result list)*
    - *Extend the tool to search for related tasks. Sometimes a task is fixed without entering a CR. Such tasks could can never be found using the tool.*
    - *Add the possibility to search within a particular period. This prevents from getting irrelevant search results from 9 years old in the result list.*

12. **Comments?**

    Developer 9: *Overall the tool is very useful and it saves me a lot of analysis time.*

Interview RDF tool Developer 17

1. **Wat was je eerste indruk van de tool? Was alles meteen duidelijk, of was het nodig om de handleiding te lezen?**
   What was your first impression of the tool? Was everything immediately clear or was it necessary to read the manual?

   Developer 17: *The tool was very clear.*

2. **Gebruik je de tool voor het zoeken van gerelateerde defecten, of voor andere dingen zoals bijvoorbeeld het zoeken naar duplicates?**
   Do you use the tool for detecting related defects, or for some other reasons such as finding duplicates?

   Developer 17: *Most of the time I use the tool for finding duplicates. Sometimes I also use it for finding related defects.*

3. **Welke zoekfunctie (met of zonder description) levert het beste resultaat op?**
   Which search algorithm (with or without description) delivers the best results?

   Developer 17: *In about 90% of the searches I use the description-based search algorithm. When a stack trace is present in the description, this algorithm is definitely better. Also in case of crashes (11504) the extended algorithm performs better.*

4. **Wat vind je van de resultaten die de tool geeft? Zit er altijd een zinvol resultaat bij, of is dit slechts sporadisch het geval? Nu worden er tien resultaten getoond, is dit voldoende?**
   What do you think of the results indicated by the tool? Is there always a meaningful result included, or is this only occassionaly the case? Ten results shown, is this sufficient?

   Developer 17: *In approximately 7 times out of 10 a relevant result is returned. Especially stack trace searches return a lot of relevant results. It is also easier than using Synergy, because you do not have to enter the keywords yourselves. An option to return more than 10 results would be nice, especially when a lot of relevant results are returned.*

5. **In de lijst van resultaten is ook de huidige status van het CR gegeven. Gebruik je deze informatie, of is deze niet relevant? Zou het bijvoorbeeld handig kunnen zijn om de lijsten per status te grouperen.**
   In the list of results also the current status of the CR is given. Do you use this information, or is it irrelevant? Could it be useful to group the result lists by their status?

   Developer 17: *This status could be useful. In my case I would focus on duplicates and resolved defects.*

6. **Wat vind je van de performance, werkt de tool snel genoeg?**
   How do you think about the performance, is the tool fast enough?

   Developer 17: *Faster than searching manually. Also a lot faster than Synergy, so the performance is good.*

7. **Heb je de functionaliteit om synonymen toe te voegen gebruikt?**
   Did you used the functionality to add synonyms?

   Developer 17: *No, I did not used this functionality yet.*

8. **Heb je de functionaliteit om accenten toe te voegen wel eens gebruikt? Zo ja, werden de resultaten hierdoor zinvoller?**
   Did you used the functionality to add accents? If so, did the results became more useful?

   Developer 17: *I used this functionality rarely. If I know what I am looking for, than I will use Synergy to search.*

9. **Heb je de functionaliteit om additionele zoekworden toe te voegen wel eens gebruikt? Zo ja, werden de resulaten hierdoor zinvoller?**
   Did you used the functionality to add additional keywords? If so, did the results became more useful?

   Developer 17: *Yes, I used it frequently. The results were getting better. However, I think some templates are still not filtered correctly.*

10. **Heb je ook de functionaliteit gebruikt om negatieve zoekworden toe te voegen? Was je van deze functionaliteit op de hoogte? Werden de resulaten hierdoor zinvoller?**
    Did you used the functionality to add negative keywords? Were you aware of this functionality? Did the results became more useful?

    Developer 17: *Seems to be useful, but I did not used this functionality yet.*

11. **Wat zou je willen verbeteren aan de tool?**
    Do you have suggestions to improve the tool?

    Developer 17:
    - *Extend the top 10 result list according to a user defined value.*
    - *Filtering on states*
    - *Opening the new defect in Synergy (was already included, but not clear to find)*
    - *Search for a fixed text. For instance, "state=bug".*

Short interview RDF tool (User 1)

Question 1: You used the tool only a couple of times, why don't you use it anymore?

User 1: *I only used it I couple of times to check whether the tool could be interesting for our tasks. However, in contrast to the other development teams our team only has a very small number of unresolved defects. Therefore, all defect are known by our team. As a consequence the RDF tool is not that useful for us.*

Short Interview RDF tool (User 2)

Question 1: You used the tool only a couple of times, why don’t you use it anymore?

User 2: *I used it once, because I knew a duplicate defect existed, but I could not find the duplicate defect. In this particular case the tool helped me and I could find the duplicate.*

Untill now, I had no particular reasons to use the tool. However, it could be useful to use the tool for every defect to check for duplicates. The performance in terms of speed is good. Applying accentuations was in the beginning a little bit unclear.

For the future I plan to use the tool more often.

F
Appendix: Management Summary

MANAGEMENT SUMMARY
stageverslag / rapport

Korte samenvatting van het verslag / rapport:

Defects, also known as bugs, are a major issue in the world of software development. Developing complex bug-free software is a major challenge. Fixing defects is very time-consuming, therefore different approaches were introduced for improving the defect fixing process. A commonly used method is duplicate defect detection. Using this method duplicate defect reports are automatically recognized and marked as duplicate in the repository. As a consequence, the number of unresolved bug reports is reduced.

However, by applying duplicate defect report detection the number of defect reports decreases, but the time to resolve a particular defect is not reduced. According to some experts at Océ, analyzing the defect report takes most of the time. Therefore, we suggest reducing the time of solving the defect by presenting a list of related defect reports. Especially for less experienced defect solvers this approach could save a lot of time.

We consider related defect reports as are already considered defect reports, which could support the defect fixer in solving a newly reported defect. A defect regarding a text alignment problem could, for instance, support the bug fixer in solving a defect concerning an image alignment problem. Because the notion of `related defect' is vague, the opinion of the developers was taken into account.

To detect related defect reports, we applied natural language processing techniques. In contrast to duplicate detection, defects occurred on other products than the new reported defect could also be related. Consequently, we only evaluated the short summary (synopsis) and description of the defect report.

We present two algorithms which compute, for a given defect, a top ten list containing related defects using the synopsis and/or description. Both algorithms are implemented in a tool called Related Defect Finder (RDF) and evaluated in a case study at Océ Technologies. To achieve a maximum performance in terms of speed, we indexed all terms of the synopses and descriptions.

RDF was able to compute a top ten result list (based on the synopsis) in less than half a second in a repository containing over more than 40,000 defect reports. Because the best performing algorithm was only based on the synopsis, we also introduced an optimized algorithm. This optimized algorithm computes the results based on a combination of the synopsis and description in less than six seconds on average.

Our experimental results show that using the synopsis-based algorithm at least one related defect in a top ten list can be detected in 76.9% of all cases. Furthermore, in 41.7% of all cases a related bug reported showed up at the first position of the result list. On average 26.4% of the defects in the top ten list is related. The optimized algorithm performed a little bit better. On average 28.4% of the defects in the top ten list was related, which is an improvement of 2%.

During an evaluation period of one month the RDF tool was used by 28 unique users. Of these 28 users, only five users used the tool more than three times. This small number of active users can be explained by the fact that most of 28 users were curious people like project managers for which the tool was not useful. The two most active users used the tool for different purposes (related and duplicate defect detecting). According to these two users the tool is a good addition in resolving defects. For the less frequent users the tool was in some cases not useful for their tasks. Other less frequent users seemed to be experts who (in most cases) do not need a list of related defects to solve a new defect.

(De schrijver/ schrijfster werd door Océ Technologies B.V. in staat gesteld een onderzoek te verrichten, dat mede aan dit rapport ten grondslag ligt.
Océ Technologies B.V. aanvaardt geen verantwoordelijkheid voor de juistheid van de in dit rapport vermelde gegevens, beschouwingen en conclusies, die geheel voor rekening van de schrijver / schrijfster komen.)