MASTER'S THESIS | LUND UNIVERSITY 2017

Measuring Semantic Distances between Software Artifacts to Consolidate Issues from the Development and the Field

Mahmoud Nasser

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2017-09

Measuring Semantic Distances between Software Artifacts to Consolidate Issues from the Development and the Field

Mahmoud Nasser
Mail@MahmoudNasser.se

June 14, 2017

Master's thesis work carried out at a Case Company and the Department of Computer Science, Lund University.

Supervisors: Elizabeth Bjarnason, Elizabeth.Bjarnason@cs.lth.se
Markus Borg, Markus.Borg@cs.lth.se

Examiner: Per Runeson, Per.Runeson@cs.lth.se

Abstract

Identifying and keeping track of different structural representations of functionally overlapping issues is important in order to keep a well maintained issue management corpus, establishing efficient and organized response ability to develop and code software patches repairing these issues and defects. This is normally achieved by manual, time-costly reviewing-processes by special teams put up to this task.

In this project we implement a tool using information retrieval technology, that intends to help these teams make better and faster qualitative assessments by providing quantitative indications in the form of similarity scores to other artifacts within a given dataset.

This approach is inspired by a paper with a similar goal, namely detecting duplicate issue reports. That study found that 60 % of all marked duplicates could be found with the corresponding implementation of this approach. Achieving similar outcomes would contribute to improved and more effective reviewing-processes.

We use the qualitative research method of informal interviews to define the semantic distance metric to implement. In the evaluation we mainly use a qualitative method to assess the accuracy of it, but verify our findings with a quantitative method. We also investigate the scalability of the tool with quantitative methods.

As a result of the limited scope of this thesis work, the tool in its current state will have limited use in a live development environment. However, we conclude that this approach has a development potential and could bring fruitful findings in the issue management and issue maintenance field if developed further upon.

Keywords: information retrieval technology, semantic distances, issue management, issue maintenance, traceability link retrieval

Acknowledgements

First and foremost I would like to thank my supervisors, Elizabeth Bjarnason and Markus Borg, for supporting me through the entirety of this master's thesis; from the defining phase of forming the idea of the thesis, into this very state of what this work has come to be. They broadened my perspective of this field, and ultimately formed this thesis by providing their helpful knowledge and important insights.

I can not express my appreciation enough for the effort they put into this work and their endless support, which has been important through out my journey working with this project, especially their moral support and encouragement during periods of dejection.

Secondly I would like to thank the Case Company and the people who made the execution of this project possible; particularly the highly competent Senior Verification Architect, who not only practically served as my primary supervisor during the implementation phase, but also as my friendly colleague and knowledge source at the Case Company.

I would also like to extend my sincere thankfulness to the Verification Department Manager at the Case Company for navigating me through the formalities at the Case Company to get me up and running with the project, but also, simply for being the kind and humble person I got to know during my days at the company. I also feel that the Coordinator at the Case Company, the initial contact who gave me the much needed initial moral support and confidence in the thesis idea during my first days at the Case Company, is warranted a proper thanking.

Last but not least, I would like to thank God for my achievements through His will, and my wonderful family; without whom, all my achievements would be rendered meaningless. They have always been my light in darkest of days; This work is dedicated to them.

Contents

Chapter 1

Introduction

In the software development industry much time and effort is put on overhead work around actual software development, particularly as a company expands in magnitude. The larger magnitude, the less effective some problems appear to be solved. One example of this overhead work could be the management of software artifacts such as issue reports. Even medium sized software development companies could experience a problem that means poorer overview of issue reports and how they relate to each other, how they relate to different customer requirements, and how they relate to actual code within a single product or a series of products. Putting a regular clean up routine in the development process is a common solution to this problem in the industry, by e.g. having regular manual peer review meetings. An example of a clean-up task among records of issues, *issue entries*, is by trying to merge two or more issue entries of the same issues, tests or requirements that are either identical or practically the same, in the sense that it could be merged into a single better documented issue entry.

Normally, a software company would prefer to keep all issue entries instead of deleting or changing them, since this eventually results in artifacts changed to the core until the individual issue entry cannot be recognized from its initial state. This is important since change-traceability is another important software engineering field [22]. This also normally happens as a result of ever changing software artifacts such as system requirements or issue reports [15][18]. In order to keep them without obstructing the efficiency of the system. The typical approach is instead to create new issue entries as different issues evolves.

To solve this layer of complexity many companies use complex relationship indications, *traceability links*, between artifacts. These traceability links must be put in place by the users that create new issue entries, and they need to keep track of existing issue entries to detect duplicates and link them to each other. These traceability links are important to have for software development organizations and help reduce development costs [8][9]. One approach is to apply a system of main issue entries and sub issue entries. Specific issues can only have one main issue-entry each in such a system, while cloned issue entries and issue entries describing different aspects of the essentially same issue as the one described in the main issue entry, *duplicate* entries, are linked to the corresponding main issue entry.

Many issue management systems also implement relations and links between different kinds of software artifacts to indicate they describe the same functionality, just only with different structures[2]. In the main versus sub issue entry approach just described, this introduces an additional layer of complexity, since this new kind of link, a *semantic traceability link*, could in practice as well be linked to a main issue entry as to a sub issue entry. In reality this implies that the semantic link also applies on the main issue entry of the issue entry that is part of such semantic relation link. But on the other hand, if such a relation is defined for a main issue entry, it is not necessarily applicable on all sub issue entries of that main issue entry.

The review process might interfere with individual traceability links between a pair of artifacts of different kinds. The interference could simply be that a former link no longer is relevant at all in the new context, or it could mean the traceability link should be changed to apply for a different pair of artifacts or that it should apply for more pairs of artifacts. This is also considered in the review process; lost and misdirected traceability links is a problem, and the importance of maintaining and recovering traceability links is widely recognized [21][12][13]. If processes to maintain traceability links are not executed properly, there are higher tendencies for project overruns and failures [15][10][12].

Sometimes these two different types of artifacts, that have a traceability link between each other, could be fundamentally different in character; one artifact in a pair could describe a product issue, while the other could be describing a test case. This introduces complexity in the decision process of how the two different artifacts actually relate, particularly when reevaluating an existing traceability link between two artifacts being altered in one way or another, since they could be describing two different things than initially intended. The introduced difference is difficult to measure, especially with data that suffer from over- or under- documentation. It could result in over interpretation of some details or overlooking important details crucial for a decision, simply because they did not exist.

We believe that a systematic quantitative measurement of this distance, if ideally accurate, will heavily reduce the time and effort put on these clean up processes by semi-automating them and possibly pave the way for future automation, since the manual approach is proved to not scale[14], and prone to errors[10][11][13]. This would mean that the process traceability link retrieval would become easier and faster to conduct resulting in time-effectiveness and greater recovery rate, thus making the software development process even more cost-effective[8].

With this thesis work we aim to investigate if it is possible to have a quantitative measurement on this difference by using a search engine as our information retrieval technology core. The problems that are faced on the ground for the Case Company hosting this project is over- or under- documentation of issues, thus artifacts, and sometimes lack of important artifact meta data; the organization needs to early detect duplicate issues and identify semantic links to other entries of the same artifact types and traceability links to entries of different artifact types already existing regardless of being specified by the submitter or not. Therefor this project focuses on the part of the process when new software issue submissions are pending review into the issue tracking system.

1.1 Research Questions

The goal of this work is to investigate the possibility to track and specifically give a quantitative indication for two artifacts of the same characteristics, namely artifacts describing issues in different contexts, with information retrieval technology.

Another question to discuss is whether we can take this approach further by enhancing it to do the same things for artifacts of different characteristics, such as related test artifacts and discuss if the tool could be developed further to capable to give the user of the tool this same quantitative support for links between artifacts of profound difference, thus a tendency of larger semantic distances.

To break this up into managable pieces we defined the following research questions for this work:

RQ1 How accurate is the implemented assessment of semantic distance?

RQ2 How can the accuracy of the assessment of semantic distance be improved?

RQ3 How does the implemented approach scale for large sets of data?

1.2 Theoretical Background

Most modern tools and solutions today involve software development. While computing technology keeps evolving into being more complex and mobile, the process of software development gets more sophisticated and advanced; This is especially true for larger organizations with software platforms shared by a large product portfolio. Software development processes get increasingly important as software systems become more complex[7]. Even more important, considering that software has a large impact on not only an increasing number of end users, but also the society's increasing reliance on digital systems.

Figure 1.1: This figure illustrates the difference between structure and function with a simple example by considering the *string* "1111"; The string "11" is structurally closer to it than "999". Though, it is obvious that the strings' function is to represent numerical values and that, functionally speaking, the *number* 999 is closer to 1111, than what 11 is.

When developing software, the process around the writing of code is as important as the code-writing itself. There are requirement engineers that account for one of these processes, making sure the product is specified in a certain way in order for developers, managers and customers to be able to take conclusions and agree on how an end product will look like. There are also teams responsible for registering issues in the code or software solution as a whole.

From a requirement engineering perspective, there are many different kinds of software artifacts. Two common types for a software project in its active life cycle are the requirement artifacts and test artifacts. These artifacts may take different forms; they could be everything from tangible software code to abstract high level documents.

Since the structure of the two artifacts are different, we say there is a *structural* distance or a *syntactic* distance, it becomes difficult to shape and construct a test artifact that is meant to describe a specific function also described in a differently structured requirement artifact. In order to somehow measure this we usually speak of the *functional* distance, or the *semantic* distance between two artifacts[2][3], see illustrated example in figure 1.1.

If a test artifact and its requirement artifact attain large semantic distance it ultimately implies that the test artifact is not really covering and completely testing the requirement artifact. We say that the *test coverage* is low. It is of interest to keep track of a requirement's test artifacts to be able to verify that the test artifacts indeed test correct functions and qualities specified in the requirement artifacts. The project addresses the field of test coverage issues towards requirement artifacts rather than the issue of test coverage for developed source code. Code development organizations face issues with complex and time consuming tasks to manually verify if their test artifacts have proper test coverage for the requirement artifacts they are meant to test.

While there are plenty of research on how to measure semantic distances between different artifacts, there is a narrow foundation of research on automatic measurement of this metric. In the rise of machine learning and text mining it has become a popular tool to solve issues like this with artificial intelligence.

1.3 Technical Background

Most software development organizations today use some kind of issue tracking system for many purposes: registering issues found internally from customers and end users but also helping developers, testers and others in the organization to keep track of status for registered issues are two examples [16].

Different factors are to be considered in order to evaluate and decide to what extent two artifacts would be related. The most obvious factor is the similarity in title and description, since most of the textual formulation of the defects is located there. Though this factor can be useful in many cases, particularly to detect duplicates, one can note that it would not be enough to find more complex relations and traceability links between artifacts, particularly if they have a considerable structural difference. Investigating meta data of the artifacts becomes important.

We use the open source search engine named Lucene in order to index defects and their meta data to evaluate the relation between different defect-artifacts. Apache Lucene is a modern, advanced text search engine framework designed for developers to use for a variety of purposes of their choice they see fit. Lucene enables us to work with the question at issue from a high-level approach since it has its own popular and well-proven text-comparison algorithms instead of putting time and effort in reinventing them. Using text-comparison is not a new approach in this field; in fact the tool developed for this project is based on a tool called ImpRec, which is made for another project by Borg et al [6].

1.4 Report Outline

The report is outlined through different chapters to facilitate the reading of the report; Each chapter brings up different aspects and phases of the thesis project, as follows:

1. **Introduction:** The introduction gradually introduces the issue at hand by presenting relevant technical information, real life issues from the field and how they have been managed and how they are managed presently. The research questions are presented in this chapter.
2. **Related Work:** This chapter brings up work by others that is critical for this thesis project. This chapter gives a better understanding of the nature of the software engineering field in practice.
3. **Case Description:** The case description focuses on our thesis project. It describes the case company's infrastructure and the data structures available, how they are created, managed and altered. This chapter will help the reader understand the practical resources and limitations we had to deal with throughout the implementation, but also evaluation of our work.

4. **Method:** The method chapter makes up the most critical part of the report; we describe our different research methods and why we use them, we describe the approach of using a search engine. We also describe how the tool is meant to be used and how we implemented several mechanisms such as score normalization and meta data field weighting. More over, the evaluation methods and procedures are described in this chapter too.

5. **Results:** The results chapter simply presents all found results, particularly results of accuracy and performance measurements. The reader will find many tables and charts here.

6. **Discussion:** This chapter discusses and interprets the results presented in the previous chapter. The validity of the results are also discussed, and therefor error sources, limitations and flaws in the used methods are discussed. In this chapter we also discuss how the execution of the project can be improved and developed upon too.

7. **Conclusions:** The conclusions chapter seals the report by drawing final answers and conclusions to the research questions and on the project as a whole.

Chapter 2

Related Work

This chapter introduces related work and projects of relevance for our own project, to give a better understanding of how this project can be linked in a greater perspective and what kind of issues and problems that this work tries to solve.

2.1 Semantic Distances

The ability to communicate a message in a way oneself imagines it to a counterpart is not always easy. Different factors may impact this ability in different ways that can be broken down in three categories; they could be related to the people involved, the context of the communication but also the used tools and means of communication.

The concept of distances fundamentally explains the communication conditions between people. Bjarnason and Sharp have, in their case study [2], identified different distance-factors similarly divided into three categories; factors could be people-related, activity-related or artifact-related.

People have different kinds of mindsets and perspectives, thus impacting how they see things differently; a senior software developer trying to explain a piece of code will need to put extra effort into explaining it to a typical student compared to his colleagues. This is because there is a *cognitive distance* between a student, that has poor or no programing skills, and a senior software developer. The student does not think like a programmer and communicating e.g. a code explanation will take effort extra effort to compensate for this distance for the message to come across.

An example of an activity-related factor is the *geographical distance*, which is one very good example, since we are talking about physical distance; to communicate a message to people physically close to oneself, i.e. in the same room, is much easier than trying to communicate the very same message to the very same people when they are at different places than oneself; geographical distances between individuals and teams slow down the communication and can cause misunderstandings.

There is also the artifact-related factors, one of which is the *semantic distance*. It describes how documents, source code and test cases can cause these communication gaps. These different artifacts have different structural composition, while they still try to "communicate" the same "message", only to different people. System requirement specifications can be used to communicate a function-design between software system architects and customers, while test specifications can be used to communicate the very same function between testers and developers.

It has been shown that distances in general have an impact on a development project's ability to communicate and coordinate efficiently [2]. This means semantic distances impact the maintainability of a project's software artifacts, thus the development of a software project as a whole, whether it is regarding low-level artifacts, i.e. code, or high-level artifacts, such as software requirement specifications. The article also talks about this, mentioning how projects with large semantic distances tend to be costly and time consuming, since it results in lack of communication. Very often this accumulates negatively through the development process leading to poorly formulated software requirements when customer and requirement engineering do not fully communicate their messages. These misunderstandings propagate, and even evolve, through the development process chain via testers and developers, and could even have originated as a misunderstanding between the end-users and the customer ordering the software system. This results in a software system that will not meet the customers' expectations, resulting in much more wasted time and larger cost than initially accounted for.

Moreover the importance of traceability and the recovery of traceability links is mentioned. Traceability links help keeping track of these structural layers in order to understand how and where each artifact is related to another.

The concepts of syntactic and semantic distances are central for this thesis work since it is defined on these terms and concepts, making Bjarnason's work important in order to grasp the general issues caused by such factors, but also to understand the specific case of our work. This will help us understand how we can define and implement a quantitative metric for this very abstract concept.

2.2 From Bugs to Decision Support

Working with large projects in complex organizations and their information landscapes creates issues for software developers regarding having a good overview of different software artifacts such as the ever growing volume of defect reports managed for a software project over the years of its life cycle. Efficient management of incoming issue reports requires successful navigation of the information landscape.

The two work tasks involved in issue management: *Issue Assignment* and *Change Impact Analysis*, are addressed in the PhD thesis by Borg[4]. Issue assignment is the early task of allocating an issue report to a development team and is important in all large software projects; change impact analysis is the subsequent activity of identifying how source code changes affect the existing software artifacts, and is particularly important for safety-critical projects.

The solution approach described by Borg seeks to support navigation among development teams and software artifacts, based on information available in historical issue reports, such as what kind of person that usually solves particular kinds of bugs, and what software artifact that is prone to be impacted by specific kinds of bugs.

While humans get overloaded by growing volumes and inflows of issue reports, clouding their judgment and increasing analysis-time to make different decisions, the used techniques benefit from these volumes by using them as training data for the learning model to develop accurate decision support for software evolution. The evaluation of the proposed tools is conducted by both replaying the historical inflow of issue reports from industry projects, and by deploying tools in real operational settings resulting in a total of more than 60 thousands issue reports involved in the studies. The results suggest that both automated issue assignment and automated change impact analysis might be useful to support navigation in large software projects.

This PhD thesis work contains five publications distributed over three parts, or *phases*, in the process of documenting bugs to utilizing this subsequent resource; The explanatory, solution and utilization phase. Three of those have been central to the cemented approach; the work of information retrieval technology[6] has been integral in forming our thesis's approach in its initial phase, but has also been central to the whole context of this thesis regarding traceability links since they are the element that establishes a consensus of a semantic relation between two linked artifacts. Understanding what makes two artifacts semantically close helps a great deal to develop a tool that can provide quantitative value on the semantic distance of two given artifacts.

The content of the change impact analysis work [17] was not at the same level of importance as the paper for information retrieval technology, but it provided something very important; the actual tool of this master thesis is based on the proof-of-concept tool *ImpRec*, that was developed as part of that paper. It was a great help for understanding how Lucene worked in general and how it should be implemented for indexing defects in particular. It also relieved us from implementing the Lucene framework from scratch. Although our

tool is based on ImpRec, it ultimately ended up having fundamentally different functionality and a heavily altered user interface to accommodate and serve our own functionality and purposes, which differs from ImpRec's original purpose and functionality. ImpRec's purpose was to analyze possible impact an artifact could have on other artifacts if changed, and this was done by utilizing a traceability link network formed of given traceability links.

2.3 Detection of Duplicated Artifacts Using NLP

The detection of duplicated artifacts is one of many issues when maintaining a corpus of issue reports. Runeson et al write about this in their paper on detecting duplicate reports[20].

This paper describes the impact of the most simple cause in issue management related problems an organization often has, the problem of having duplicated defect reports. This creates unnecessary redundancies and clutter making the defect corpus of a company difficult, and therefore costly to maintain and work with.

Automating the detection of duplicate defect reports has proved to be difficult since issue reports are often formulated in natural language. The simple approaches used to help detecting duplicated artifacts are based on using algorithms doing, more or less, simple text-comparisons. Implementations based on this approach are prone to letting through a fair share of false-negatives, i.e letting many duplicates remain undetected.

The paper describes an approach to detect such duplicates using natural language processing (NLP) technology in which the algorithms are aware of relevant linguistic traits of natural language, making it smarter in understanding meaning. It can classify words by part of speech and map relations and references in texts to understand context, rather than simply considering the actual string. This enables for more sophisticated algorithms than simple string comparison ones.

This thesis work is related to this paper since we use a search engine implementing algorithms typically associated with implementations targeted at solving natural language problems.

2.4 Automating Requirements Satisfaction Assessment

Software system functions can be described in different ways depending on the context in which this description is needed. The different contexts usually consist of different software development processes in a software development process-chain, such as defining and documenting system requirements for different functions, creating tests for these functions and also writing the function software.

A hypothetical function will be described in at least three different kinds of artifacts; some code, a test-case (or more) and a documented requirement. While each artifact takes different shape and form, they all describe the same function, and therefore they all have traceability links between each other.

It may seem to be alright to keep those separated and independent of each other, but in order to have good overview of a software project one has to be able to verify different development process transitions, to be sure that the very same functional requirement in the last step of the process chain can be backtracked through each process in the process chain. The test artifact need to have minimal semantic distance to the requirement artifact to ensure it tests the functionality intended. A non existing semantic distance in theory means that they describe the exact same functionality.

As mentioned and described by Huffman Hayes et al [1], software development organizations periodically arrange verification processes in order to answer the question of whether the high-level system requirement artifacts are satisfied by their low-level linked test and code artifacts. This activity is called satisfaction assessment, and is useful because it helps find issues with non-satisfied requirements early throughout a project's progression, when such issues are cheap to address.

The relation of this paper to our thesis work may not be apparent, but a "system requirement" is a broad term, which in our case could be replaced by our own "issue report", thus describing satisfaction assessment in our context. This project focuses mainly on issue artifacts, but in the broader scope of things, these issue artifacts have traceability links to test artifacts. We do not lay our hands on test-related artifacts, though we mention them in the case description chapter for the reason of their importance in the broad perspective.

Chapter 3

Case Description

The Case Company at which our project was executed and whose data we utilized, during implementation and evaluation, is an international company that develops smart phones and tablets with a custom built Android version. The Case Company maintains a large product portfolio and has sold a couple of tens of million smart phone units during the fiscal year of 2015. This project has been hosted by one of the Case Company's main software development sites, and while being an important site in software development, it is not the only site for the company, as the company has a couple more sites worldwide. The Case Company's software developers collaborate with many phone service providers worldwide as every one of these providers release phones with their own branding and custom firmware.

For a company like this, issue tracking is of significant importance for managing their products efficiently. An extensive infrastructure is used to keep track of different kinds of issues, software and hardware issues alike, and different departments and teams, that have different disciplines and fields, collaborate via this infrastructure. This kind of infrastructure is usually referred to as an issue tracking system.

This project focuses on the part when new software issue submissions are pending registration into the issue tracking system. The problems that are faced is over- or under- documentation and sometimes lack of important meta-data; the organization needs to detect duplicate issues early and also need to identify traceability links to other issues already existing regardless of being specified by the submitter or not. Of course, the scenario

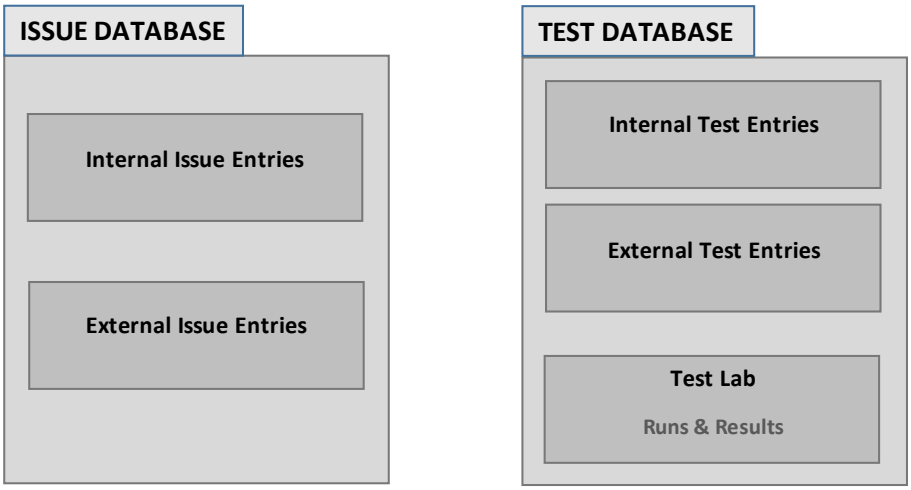


Figure 3.1: Visualization of two databases that are part of the Case Company's software artifact infrastructure. To the left is the issue database, central for this work. To the right is the test database, which is good to have in mind, and could be of interest in case a future project picks up this thesis with a broader scope, considering its close and interesting relation to the issue database.

for these kinds of problems could be encountered as a post-submission scenario. Post-submission scenarios means other employees and teams might be put in the contexts of detecting duplicates or other misrepresented data that inflicts problems that need to be resolved; the project considers some of these scenarios too.

This chapter describes how the infrastructure of the issue system works, defines and describes the cases we are interested in and form an abstraction of the issue-system to scale off unnecessary details for this project's scope.

3.1 The Case Company's Infrastructure

The Case Company's infrastructure consists of many databases for managing different kinds of software artifacts. For this project we chose to only study two of the company's many different databases to track software development artifacts related to issue report management and test case management for their different products' firmware and applications; an issue database and a test database, as illustrated in figure 3.1.

To form and implement the semantic distances metric, we made the choice to work with only one of those databases; the database central for the scope of this project, is the issue-focused database. It contains two different kinds of artifacts of relevance. Issues reported by operators, mainly during acceptance testing, are submitted as the first kind of these

two artifact types, as *customer acceptance entries*; to simplify the terminology we will be calling them *external issue entries*. The other, more common issue artifact type, are the entries created and submitted by the Case Company employees themselves as they find issues to report and are referred to as *internal issue entries*.

The internal issue artifacts and the external issue artifacts are not only different by name and the context they are composed in; they are distinguishable by the meta data fields available for each of the two. The internal issue artifacts have a larger variety of meta data fields used internally by the Case Company, while the external operator issue artifacts have a limited variety. Naturally there are some common meta data fields such as submission date, description and so on.

The test database that holds test case artifacts and test logs has some relevance to this project; even though the artifacts it contains do not have any impact on the tool nor this specific project, the tool could be modified to apply in a larger scale and include the test database. By understanding how they relate in the larger perspective, we think it will give a better understanding on how the rather simplified scope is relevant to the industry.

Similarly to the issue database, the test database holds different kinds of artifacts; there are the external *operator test cases* from a large set of cell service providers, or simply *operators*, worldwide; and there are also *internal test cases* created by the company employees. These provide complementary subsets of test cases to the operator test cases, ensuring higher coverage and concludes redundant, sometimes duplicated, test cases from the different operators. The test database also contains *test data* artifacts, containing fail status for the last runs of all the test cases and information about traceability links between test-artifacts in the test database and issue-artifacts from the issue database.

3.2 Software Artifact Dependencies and Management

The issue database consists of different types of entries, two of which are the internal issue artifacts and external issue artifacts. The external artifacts are issue entries created when discovered during customer acceptance tests and are often created by employees at the operator companies.

Since the database has two types of issues, this makes the collection of external issue entries prone for duplicates if there e.g. is a generic defect that reproduces for more than one operator. These issues could be described differently and the documentation quality could vary between them.

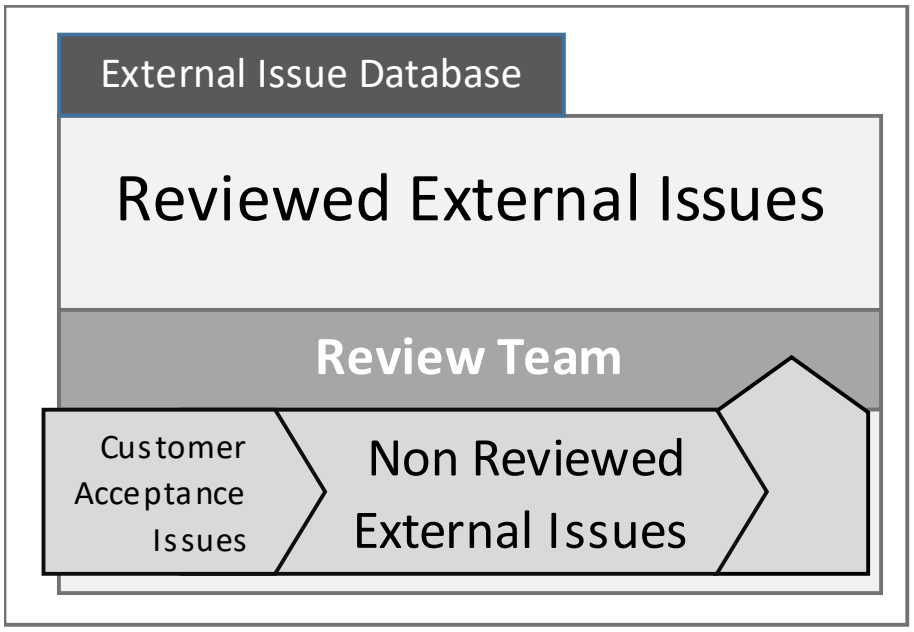


Figure 3.2: This figure illustrates the process of external issue submissions. An external issue originates from from customer acceptance tests and are submitted by different operators' employees. All submitted external issue artifacts awaits review by the Case Company to be granted a status and proceed, or to be rejected for various reasons.

Each external issue submitted must be reviewed by one of the appointed review-teams at the company. When such a team evaluates a submitted entry, they will make decisions on it, such as deciding the priority of the issue, if they decide it in fact is an actual issue in the first place. The submission teams are responsible for searching for possible duplicates and traceability links among existing issue artifacts that could already be describing the very same issue. This procedure is illustrated in figure 3.2.

If the external issue entry submission is deemed an actual and unique issue, an internal issue artifact is created, based on it. Internal issue artifact-specific meta data is added, if available for the individual or team creating the internal issue artifact. If it is deemed not to be an issue it is rejected and no internal issue artifact will be created of it, or it is linked to an existing external issue if it is found to be a duplicate.

Entries of the internal issue artifact type are always submitted by employees at the company. Internal issue artifacts are not subject of review, as the case is for external issue artifacts. While internal issue artifacts are submitted by employees at the Case Company, their documentation quality can vary as they can have different origins, as shown in figure 3.3: Internal test issues can be submitted by different employees from different departments and units; internal issue entries could be derived from reviewed external issues, originally written by employees at the operator companies and it could be written by end users if collected from online support-forums as an example.

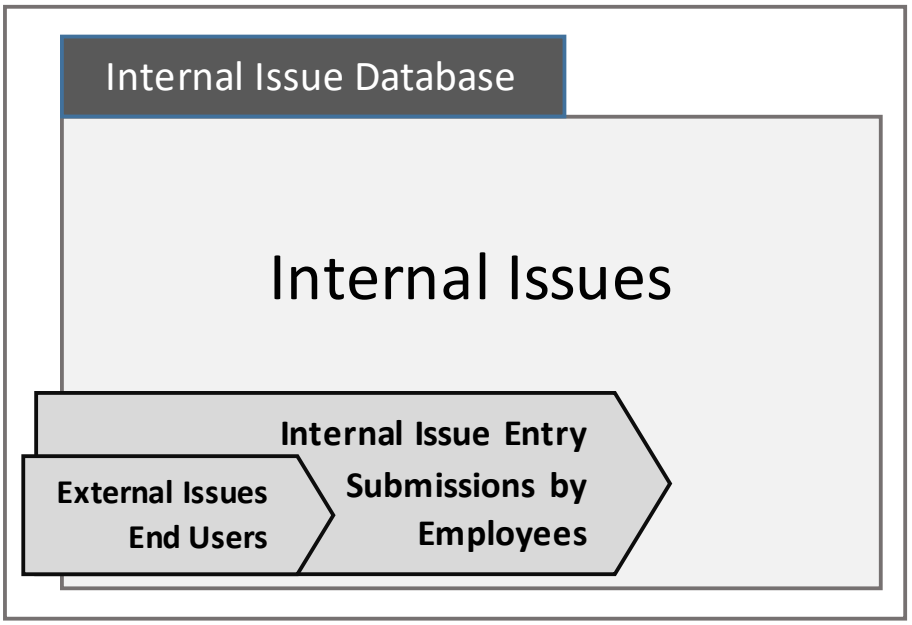


Figure 3.3: This figure illustrates the submission process of internal issues. An internal issue can originate from different sources. All submissions of internal issue artifacts are done by company employees on behalf of themselves, as duplicates of external issue entries or defects reported from end users.

The internal issue entry can have many origins. If it originates as a duplicate of an external issue, the description of it is seldom modified. However a lot of meta data that is not represented in the same way, if at all, within an external issue artifact can be appended.

Each external issue artifact is an entry created by operators and therefore each external issue entry is duplicated as an internal issue entry, if it is not rejected by a team that handles and reviews incoming external issue entries. Each internal issue entry created as a duplicate of an external issue entry is therefore linked to its duplicate external issue. Other Internal issue entries are on the other hand created independently, and are often created this way.

This creates a relation between the internal issue entry and the external issue entry it was duplicated from. This is a relation that is documented within a meta data field of the internal issue.

Other cases where internal issue entries will be submitted into the system is the obvious one; whenever someone at the Case Company discovers an issue a report will be filed for a responsible team to look into and eventually submit. This is also what happens when an end-user reports an issue.

Internal issue entries themselves have an additional layer of hierarchy. An actual issue can be described by two or more different internal issuse entries. One of these entries needs to be identified as a master entry for the issue, as illustrated in figure 3.4.

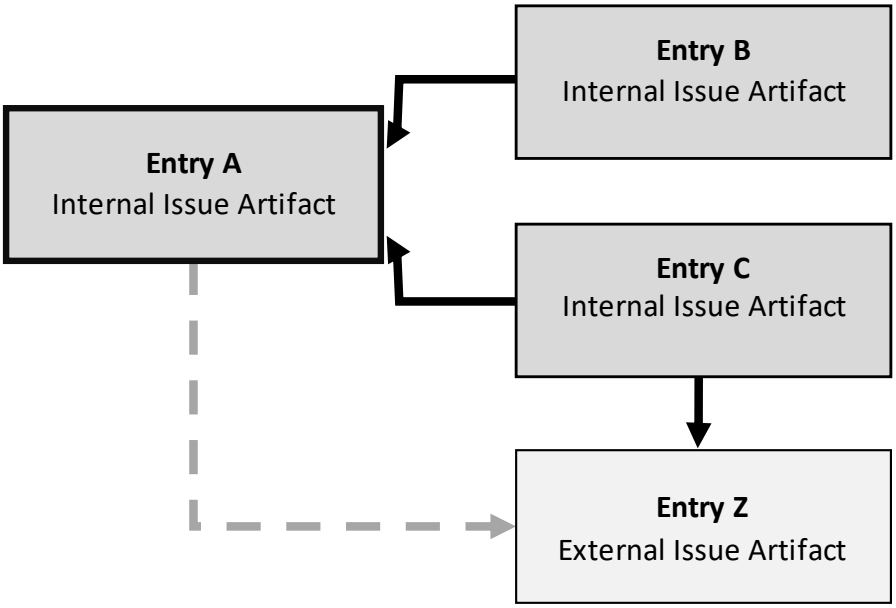


Figure 3.4: This illustration consists of an example set of relations between four different issue artifacts. One can see that entry C is in fact submitted as a result of the external issue entry Z. Internal issue entries B and C are duplicate entries describing the same issue first described in internal issue entry A, making that entry a master entry. The dashed relation shows that, in some regard, the external issue Z is also linked to the master internal issue entry A.

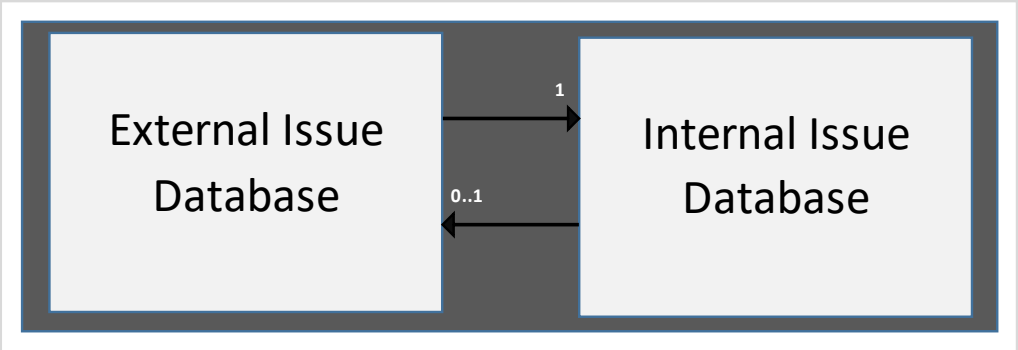


Figure 3.5: Visualization of the issue database infrastructure abstraction where internal and external issue entries are considered to be part of two separate databases by the tool. We always expect an external issue entry to have a traceability link, a duplicate, among the internal issue entries.

Technically speaking, a master internal issue entry is an internal issue entry *not* linked to another internal issue entry. As a result; all internal issue entries without a link to a master internal issue entry is considered itself a master issue entry, while those who have such a link is considered a duplicate to the master internal issue entry it is linked to.

Moreover there are database-transcending links worth mentioning. The internal issue artifact can have a link to multiple test case entries, both internal test entries as external test entries. There are also flags within the internal issue artifacts linked to the test lab in the test database to keep track of issues that have been solved. Also, naturally, the test lab has links to internal and external test case entries to be able to log them and their latest execution results.

3.3 Project Constraints and Scope Abstractions

The issue database will be the only used infrastructure during this project and its implementation, M-Tool, thus only the data dependencies and data flow within the issue database will be studied. The infrastructure will therefore be regarded in a scaled down manner.

The internal issue entries in the tool will be considered to be entries in an "internal issue database" and the external issue entries will be regarded as part of a separate "external issue database", as illustrated in Figure 3.5. This is also the case for how the tool will regard these different artifacts to simplify parsing and indexation matters. The tool will expect all external issues to have traceability links to their respective internal issues, despite the fact that this is not the case for all external issues, since some of them are rejected in the review process.

As briefly mentioned, when an end-user issue is submitted to the Case Company, an analysis of it needs to be conducted to find out if the issue already is known and reported e.g. by internal resources or operators. It is also of interest to use this data in another way; When a new product is being evaluated internally or by an operator and an issue is found, it is of interest to know if this issue is reported in other products too.

3.4 Uses for M-Tool at the Case Company

The Case Company could use M-Tool in a variety of contexts for different purposes. Mostly it could be a help at the reviewing process of new issue submissions since it is critical for the company to acquire links and in overall to map the structures and relations of the new entry to existing artifacts.

Two of the common uses of the tool at the company that we focus on with this work is described in scenario-form; the scenario of reviewing newly submitted external issue artifacts, and the scenario of attempting to retrieve traceability links between external and internal issue artifacts and also to retrieve semantic links among internal issue artifacts.

3.4.1 Scenario 1: Reviewing Submitted External Issues Pending Review

A typical scenario where the tool might make itself useful at the Case Company is an all day process at the entry review team. The team uses the new external issue artifact entry to conduct a search among the existing entries in the relevant dataset to find similar and relevant entries. It is of high value to early find duplicates and relations to other artifact entries for this new external issue in order for it to be registered in a way that will be useful for the organization as a whole, but since this normally takes time, the infrastructure suffers from under-documentation with many missing links between artifacts.

Formally, the review exclusively happens on external issue artifacts. However developer assigned internal issue artifacts to solve can apply this scenario to do an informal review of their own for internal issue artifacts too.

3.4.2 Scenario 2: Tracking Artifact Relation to Other Artifacts

During the development of specific software components the Case Company employees might encounter confusion when trying to get an overview of the status for a component if they keep stumbling upon two or more different issue artifacts seemingly describing the same issue, but have no apparent link according to the system.

This could be a sign of a widespread lack of accurate documentation between artifacts in a specific dataset. It is of interest to be able to track the efficiency of the issue reporting system and to be able to identify problems in it. One metric of value is to detect duplicates and to identify possible relations between entries with no recorded relation, a semantic link. The tool can here be useful to find duplicates and possibly link them to a master issue.

Chapter 4

Method

In this chapter we present our method of research and the approach of the measurement assessment implemented. To explain the approach we first properly introduce M-Tool and how it is intended to work from a high-level perspective before actually diving into the technical details of the approach. The evaluation methods are also described in this chapter.

4.1 Research Method

The procedure of this work is best described with a method three steps, or phases, as follows:

- First an initial definition of some kind for the measurement of semantic distances between software artifacts need to be defined and established, at least as a starting point.

- Based on the definition in step one, we develop a tool that quantitatively decides the semantic distance between two linked artifacts and gives some kind of indication of degree of similarity to enable the user of the tool to do qualitative assessments. It should also suggests or hint the user on possible links within two unlinked but semantically non-distant pair of artifacts, even though this was not developed as an independent mechanism.

- Finally we evaluate the defined measurement by evaluating the developed tool, and decide whether it is useful and to what extent, if that is the case. We also discuss how the measurement can be enhanced and for what purpose. This evaluation is also done by informal interviews with our Case Company supervisor, a Senior Verification Architect, and is also backed with quantitative data collected by accuracy measurements on preset prepared searches.

Of course these three steps are not strictly procedural, and the second implementation step in particular is heavily iterative and closely related to the first step of defining the measurement. The implementation step is also special in the way that much of the practical work evolves around it and in this step we will produce the prototypes of the tool. This step is also crucial in order to produce data and acquire knowledge in order to answer the research questions.

The data collection methods used for this work are many; informal interviews with different people with knowledge of the project and dataset cases used throughout the project were deemed crucial in order to initiate and set the project's practical scope in order to proceed with the project to a workable stage. While the interviews were heavily conducted in the initial phase, they arose throughout the project, often spontaneously as we felt the expertise knowledge were needed to make informed decisions. The interviews were executed in an informal, casual and verbal-only manner.

We also performed studies of cases with qualified employees, mostly with our supervisor at the Case Company, a Senior Verification Architect, for choosing relevant datasets, which is important since we are to be considered as outsiders when entering the Case Company for the first time for this thesis work. Different data for different projects, or time periods or even portions within a subset of data could be of different qualities and characteristics. Other examples of when the case study methods are applied were when the statistical execution evaluation was performed to measure performance and accuracy of the tool[19].

4.2 Solution Approach: M-Tool

M-tool, the name of our implementation, is based on another tool written by Borg [17] called Imprec. The implementation structure of the Lucene framework was adopted and slightly modified to accommodate our own data structures and technical needs. Regarding M-tool's graphical user interface, while we based it Imprec's graphical user interface, we turned out with a very different layout and structure. The rest of M-tool is, to an large extent rewritten, extended and re-factored.

We analyze the artifacts' structure and their meta data in the infrastructure of the Case Company, to map every aspect of it to individual, semantic-representing, fields in the search engine. We also need to decide how different fields of the different types of artifacts relate to each other. In our case this was not complicated; since we use issue artifacts, an artifact already has different kinds of meta data describing different dimensions of an issue,

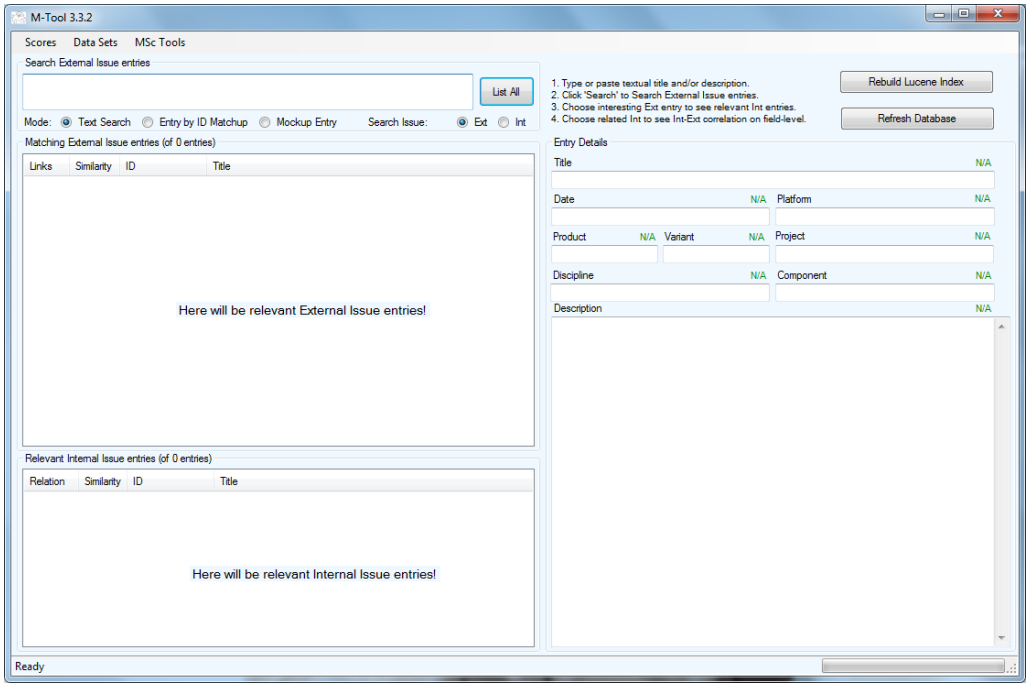


Figure 4.1: M-Tool in its untouched initial view. The layout is divided in four areas; search-options, search-results, Artifact Details and an area with general instructions and global operation-buttons

e.g. version number, project, date etc, besides the issue itself. Since both artifact types are similar, and both describe an issue defect, it is also easy to map these different represented dimensions of an issue to each other; both artifacts have a meta data field representing e.g. the software project and the software version number for which an issue was detected.

It is important to keep in mind that the semantic distance measurement in this work has an abstract definition, since we define it on a rather high abstraction level when using a search engine's own methods and algorithms. Instead we can speak of the definition in terms of how we process the artifacts through the search engine, as opposed to e.g. mathematical formulas. Different algorithms and mechanisms of the tool can be linked to the functionality and layout of the tool. The layout consists of a window with four main areas and surrounded by a tool-bar at the top and a status bar at the bottom, as shown in figure 4.1.

The different areas have different functionality in focus; the top-left area is for setting up a search, the bottom-left area is for examining the search results, with the possibility to perform a search by artifact for the other type of artifact than the one first used.

There is also the bottom-right area, a significantly larger area under the top right one, used to view meta-data values for different meta data fields of an artifact. The top-right area has short instructions of how to use the tool and two buttons to retrieve and store new data locally or to rebuild the index of the locally stored data with new settings and parameters.

4.3 Using M-Tool: Preparing the Tool for Use

If the tool is used for the first time, or if the user of the tool wants to fetch a new dataset, the upper-right area of the user interface is the place to begin. From there we let the user make the tool fetch the preselected dataset to read and parse them into the tools data structures; for short this process is called a *Database Refresh*. This action will also automatically build a local index for the search engine in order to be ready to execute searches.

From this corner of the user interface we also let the user manually rebuild the index for the search engine. This is necessary when the user modifies the search engine parameters.

The database refresh action locks the tool to set up a network connection to the company infrastructure for the issue database and queries it with a chosen dataset query. The tool awaits the server to execute the query and to respond with the data. When the server transmits the result the tool receives it and parses the data to store it in the application-specific data structures.

The dataset query is actually a pair of dataset queries; one is specific to internal issue artifacts and the second is specific to external issue artifacts.

4.4 Using M-Tool: Setting Search Parameters

Depending on the scenario for the context and the use case of the tool one would want to use search for different things, or execute a search differently.

We let the user choose between the two issue artifacts to list in the primary result list view user interface element. The user can also conduct a search in different ways by altering the search mode. The intended use is to search for an artifact and list it in the primary search box, to then allow the user to search by one of the listed results to find relevant entries from the other kind of artifact. For instance, we could want to check if a pending external issue entry, regarding a connectivity issue, has been duplicated as an internal issue entry. We set the tool to search by external issues and perform e.g. a free-text search on the external issue entry's title to find it in the tool. When found, we select it to perform a search with it, to check if the tool can find an internal issue entry that it deems similar to it.

When searching for the external issue entry, one would want to conduct this search in a different way, of the three different ways the tool offers; search by an entry mock up setup, search by a free-text string and an entry matching by ID search.

The mock up entry search mode allows the user to design an advanced search by letting the user decide and set the values of meta data fields for the search engine to try and match against the other entries and their corresponding values. More importantly this allows the user to choose to leave certain meta data fields empty if they are not vital for the specific case and need, thus making the tool ignore these fields instead of negatively impacting potential matches' scores simply because these same meta data fields does not match.

The free text search mode on the other hand is technically a mock up search, setting the meta data fields for the mock up with the inputed string typed by the user. Therefor this search mode is the less suitable if the user seeks a very specific set of results, but very suitable to do more general searches where the indication of the scores are not important.

The entry matching method requires the user to be very specific; the user needs to put a valid entry ID for an existing entry in the tool's current indexed dataset, but the user can also put a globally valid ID, meaning it exists in the Case Company's infrastructure and is available when the infrastructure servers are queried for it. The tool takes this entry and searches with it as input, to find other entries similar to it. If the entry is not part of the locally stored and indexed dataset, the tool will perform an online search by sending a query, specifically constructed for the inputed entry ID, to the infrastructure servers. What technically happens if the entry is found and fetched, is that the meta data fields of the entry are used as values in a mock up setup.

The user of the tool can also choose to do an empty search, resulting in the tool listing all entries of the set issue artifact. This is a function for general browsing of the current stored and indexed dataset and can be useful to do a simple sweep of a newly refreshed or indexed dataset.

4.5 Using M-Tool: Interpreting the Search Results

When a search is done and the results are presented, there are different ways the user of the tool can look at the result to make qualitative decisions. If anything, the first thing the user will notice is the order of the results; the order of the results can be considered as the tool's rank-order of the most similar listed first. The rank-order is determined by each search result item's similarity score, which leads us to the other factor one can examine; a similarity score is something the search engine determines internally, of course influenced by the settings and parameters we as authors of the implementation have set, but also what we allow the user to set. The similarity scores can be very general, but the user has the possibility to view each issue artifact's meta data fields and study the scores to understand why a search result entry got the similarity score the way it did by looking at and comparing the field to field scores. This whole process can then be repeated until the user is satisfied with the overall result, as illustrated in figure 4.2.

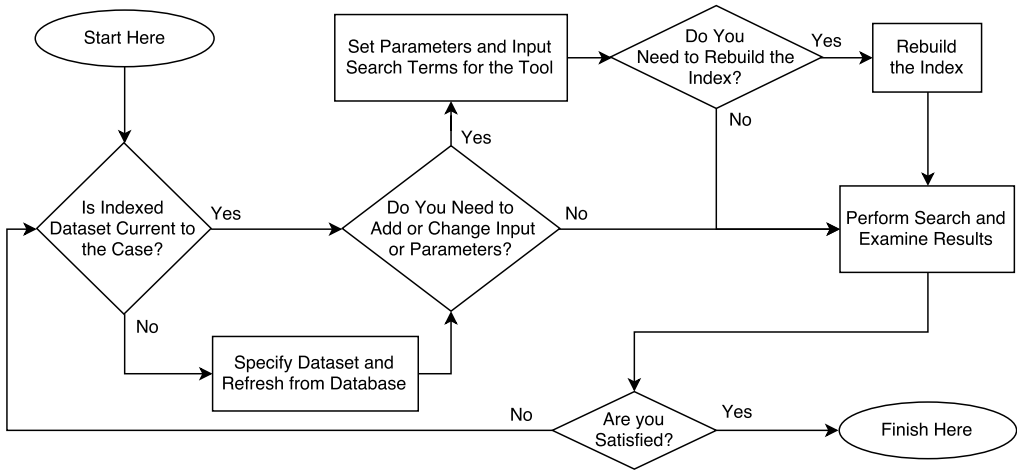


Figure 4.2: This figure illustrates the intended way of using M-Tool in a systematic way.

4.6 Understanding the Search Engine

When executing a search many things happen in the background before the results are complete and ready to be presented. The tool operates at a high level thanks to the use of the Lucene search engine library.

A search engine is a software framework designed to index structures of data, called *documents* in search terminology. A document can have different meta data fields, attributes, depending on what it represents. It can then receive external search queries to try and match with the indexed documents and returns a list of documents and their ranked similarity score to the search terms.

If the case for this project each indexed document represents an issue artifact. The attributes used are the issue artifacts' meta data fields Issue ID, Title, Date, Platform, Product, Product Variant, Project, Discipline, Component and finally Description. Some of these attributes are produced by splitting or combining actual database meta data fields (table-columns); this adds some complexity in understanding how different meta data are connected and is, as we experienced, crucial to understand in order to build a helpful set of meta data fields to utilize.

Lucene is used as the search engine core in this project tool implementation, M-Tool, in order to index and query the artifacts. Lucene is an open source engine used to index text and perform analyses to it in order to make it searchable. The different meta data fields is inserted into standard Lucene string fields. Lucene is first and foremost a string-based

search engine. It does support some other types of fields, such as numerical fields. However our tool exclusively uses string fields for indexation, even for meta-data of numerical nature; this choice was made as a result of the fact that we noticed the existence of this feature mid-project, when it would have cost us much time to implement.

Lucene supports the use of so called *boosting* for different meta data fields. By default all fields are treated equally, but the use of boosting can be useful if particular meta data of a Lucene document should impact the total similarity score of two documents more if it matches with another document's corresponding meta data field or fields.

As an illustrative example of the usefulness of the boost-feature in search engines; imagine a person being in the mood to eat. This person decides he wants to eat at a certain well known hamburger restaurant chain and uses a modern, general purpose, online search engine and types "McAdams". The search engine will most likely show nearby "McAdams" restaurants, instead of simply showing the most popular "McAdams" restaurant that is indexed by the search engine, in a far away place, just because it goes by the same name. This is because the search engine considers other meta data, such as the users location, or the nature, or call it category, of the search, being *restaurants*.

However, it is interesting what the search engine would show if this person is abroad in a country where there is no McAdam restaurants. Some search engines would probably show information about the McAdam company, or even yet, the closest McAdams restaurant anyway, even if that meant the person would need to take a flight there. If the search engine is smart enough however, it would boost the search by the *location* and *category* meta data fields, yielding a higher score for indexed documents that have similar values for the location and the category meta data fields, as the search instance's values for the same meta data fields. A search boosted this way will instead be helpful to the hungry searcher and show them nearby and local alternatives to McAdam restaurants.

So bringing this example back to our case, we can conclude that some meta data fields should understandably yield different impact on the similarity score. For instance, if there are two issue entries, for which the actual issue description is very different, despite the fact that they are related by many other factors, the user would expect the tool to list these in the result as being to be similar. Without boosting, the longer string of the issue description would yield a higher impact on the similarity score. This would happen when e.g. comparing the first issue entry with a third issue entry that has a very similar issue description as the first issue entry, even if the third issue entry does not belong to e.g. neither the same project nor the same component of the phone and software, as the first two do.

One approach could be to set the boost for the different meta data fields to fix values. That would be a nice overall solution if implemented for a search engine with a large dataset that slowly grows and where the overall indexed documents' characteristics never drastically change in nature. In the case of this tool however, the user changes between different, often distinct, datasets of different nature and character. There needs to be a scheme for each dataset in order to efficiently identify valuable differences or similarities that could be of different value for different kinds of datasets.

When looking into a particular dataset, where a majority of the entries are similar in some regard, different meta data fields will in reality yield a larger impact than they would be expected to do in larger, more diverse datasets. For instance, the issue entries in the particular dataset could all be part of the same product. This would then make the product of the issue a less significant factor for that particular dataset, and therefor make it appropriate to apply an accordingly smaller boost on the meta data field describing the product.

4.7 Score Normalization

Similarity scores generated by Lucene is difficult to read and measure without prior knowledge in how the engine works and without a greater overview of the currently indexed dataset. By requiring from the user to know how the search engine works and have a more detailed overview of the current dataset, we destroy the purpose of this tool since it is meant to give that overview we are requiring; to solve this matter, the tool needs to normalize the score over a fix range, to make the score intuitive for the users to interpret.

The tool normalizes the Lucene score to a fixed score-value range of zero to one hundered, $NormalizedScore \in [0, 100]$, with the advantage of being able to view the normalized score as a value in percentage units. This is done by taking the current software artifact, *Artifact x*, and compare it to itself to get the absolute similarity score $AbsoluteScore_{x \to x}$. Since $AbsoluteScore_{x \to x}$ is the highest absolute score possible among any artifact comparisons to *Artifact x*, we assume the maximum absolute score be the equivalent of the maximum normalized score; the value of 100. This is logically straight forward too, since a comparison with an identical artifact entry should yield the highest score possible.

When other artifacts, e.g. *Artifact y*, is compared to *Artifact x*, their absolute score $AbsoluteScore_{x \to y}$ will be normalized against the maximum absolute score $AbsoluteScore_{x \to x}$, by the following formula:

$$NormalizedScore_{x \to y} = 100 \times \frac{AbsoluteScore_{x \to y}}{AbsoluteScore_{x \to x}}$$

The individual field-to-field scores are similarly normalized too. The accumulated sum of all field scores for a current pair of compared artifacts is used as an index to normalize these individual scores around it, thus distributing a *percentage share* among all fields.

4.8 Field Weight Tuning

Initially the weights were set to arbitrary values given as a matter of personal decision. The values were then fine-tuned based on a couple of artifacts that needed to score high against each other, based on different meta data fields.

When trying to make this tuning procedure less arbitrary and more systematic as prescribed in Borg's work[5], some quantitative metrics were considered; one of these quantitative metrics was to count all different values for each meta data field for a dataset, and set a higher weight for a meta data field that accounts for a higher share of the total count.

This quantitative method of setting a weight was chosen since it made sense to give a field with minimal variation within a given dataset less impact on the similarity score, while giving higher value-variation within the same dataset a larger impact on the similarity score.

A problem with this is that the concept of weighting treats all involved meta data fields as equals; this is problematic since different meta data fields could be very different in nature. As an example, imagine the entire product catalog at the Case Company would be included in a dataset, so that we can take a look at all values the meta data field, e.g. "product", can have for all the included issue entries. The variation of the values for that meta data field would stand small in front of the value-variation for the meta data field of e.g "description", that could include a proportionally large variation. In this example, and with the given approach in general, the "description" meta data field would get tremendous weight compared to the "product" meta data field, which would make the tool give the "description" meta data field such importance, that all other meta data field values would practically not be considered to decide the similarity score.

To solve this dilemma there needed to be an element of manual tuning without the arbitrary decisions; This was solved by qualitatively producing six different datasets collaborating intensively with senior employees at the Case Company providing the data. The first four datasets where produced first in order to, as far as possible, eliminate biased results for the later evaluation-procedure.

Basically the meta data field weights were set up according to the quantitative method based on value-variations and were then adjusted to fit expected reference artifacts for internal issues listed with external issues, but also duplicates among internal issues and duplicates among external issues. The last two datasets were constructed when the project reached a more mature phase to primarily evaluate two of these tuning's efficiency, but also to have a wider array of datasets to do a more absolute measuring of the tool's accuracy in the final phase, based on master entries.

4.9 Dataset Setup

To be able to setup datasets that are helpful when refining the implementation we need to mainly consider a variation of two factors: Size of the dataset and characteristics of the issue description.

In order to do that we defined a total of six datasets, summarized in table 4.1. Where as of our case, four of them were defined first when the tool reached a stage of structural maturity, meaning the tool's purpose was established and mainly set. The four datasets were then used to tune and try to find optimal metrics and factors to guide us in setting indexation parameters, namely the boost settings.

The last two datasets were defined when the tool became mature in function and began entering a stage were evaluation of it appeared to be of relevance. Concretely speaking this occurred for us when we decided that there were no more optimization with the set structures to improve accuracy, as subjectively perceived.

Datasets four and six are strictly small subsets of internal issue entries with an external issue entry link, and their linked external issue entries included.

The six datasets are defined as follows:

1. Dataset 1 consisted of a large set of data from a specific product and time-period with all kind of software issues related to the project, from simple applications to hardware-related software.

2. Dataset 2 is a subset of Dataset 1, but is limited to specific software components of different nature. Eight of those components are of a high-level nature and relate to a phone's basic applications such as the contacts and messaging applications. These high-level components makes up for 70 % of the dataset while the rest of the dataset consists of two other software components related to code that handles connectivity hardware such as NFC and Bluetooth.

3. Dataset 3 is also a subset of dataset 1, but distinct from dataset 2. Dataset 3 is almost the same size of dataset 2 and only consists of internal issue entries of four hardware related software components.

4. Dataset 4 is a subset of Dataset 3, but only includes internal issues that are not master entries and that have an external issue link, i.e. the internal issue entries stems from external submission from operators. Dataset 4 was designed to test score and ranking accuracy for duplicated entries and score and ranking accuracy for internal issues linked with external issues.

5. Dataset 5 has much characteristics in common with those of dataset 2, in terms of the composition of the software component types it includes. Though the included entries applies for a different product, making the two datasets unrelated.

6. Dataset 6 is designed for evaluating duplicated entries and accuracy for internal issues linked to external issues and therefore only includes a limited amount of entries, as for dataset 4, but sets of entries for a different product.

Dataset	Internal Issues [Count]	Master Records [Count] ([%])	External Issues [Count]
1	5 468	4 495 (83%)	681
2	761	669 (88%)	120
3	832	715 (86%)	832
4	38	0 (0%)	38
5	1 763	1 520 (86%)	210
6	39	0 (0%)	39

Table 4.1: The table shows the composition of the different datasets to give an overview of their characteristics.

4.10 Evaluating the Accuracy

To determine how accurate the tool is, we need to have some kind of a reference point. A reference point is useful because it sets a standard of expected result for an evaluation method. An ideal kind of reference point for the application would be expected entries in the search results with expected similarity score and ranking based on data that the tool does not account for. That kind of specific data does not exist, but fortunately there were meta data that describe relations between internal issue entries to each other. An internal issue entry could be a master entry for a specific issue, or it could be a duplicate and hence linked to the master entry.

The tool is completely blind for master entry links between internal issues. For this we used an internal issue connected to a master entry to find out how well the tool ranks it's master entry (if at all). So to measure the accuracy we exported a spreadsheet listing all non-master internal issues entries and tested around at least two prepared search sets for each dataset, with two to five different search set ups within each prepared search set.

There are three types of searches included within each prepared search set:

1. A search by the linked external issue entry, to find its corresponding internal issue entry, given with a traceability link.

2. The internal issue non-master entry was used to do a search in the dataset, to find the corresponding internal issue master entry.

3. Along the same search as the second, score and ranking data for *other* internal issue non-master entries linked to the same internal issue master entry were collected.

4.11 Evaluation of Performance and Scalability

In order to determine whether the tool is scalable we decided to do performance testing on different procedures or *operations*. The chosen operations are mostly straight forward; the indexation and searching operations are central to the tool and the execution time is obviously dependent on the amount of indexed data. But less obvious, yet important operations are the local database refresh, which means the tool need to connect to a remote server, query it twice, receive data and parse that data.

Since the connection-part of the operation is mostly arbitrary and dependent of external factors such as the computers network connection, the server's workload at the time of query and so on, this metric is measured twice: a complete full refresh measure, which is the time it takes from pressing the "Refresh Database" button until the tool has indexed the received and parsed data; But also the time it takes to connect and send a query to the server, and to receive the result data from the server.

The measuring was done by using a time-secure method in C#, named *Stopwatch*. Since the actual time results themselves were not deemed as an important metric for the performance and scalability, the measure hooks were not placed with great thought. Of course common sense was applied and it was made sure that these hooks were placed consistently.

In addition, two more operations were used to measure very simple operations: listing all indexed internal issue entries, and listing all indexed external issue entries respectively.

Each sample was measured programmatically and triggered with the action measured itself within the user interface. Each action measurement was conducted on four datasets different from the previously defined datasets. Each type of operation had a different minimum number of samples to be taken, so called sample count, as shown in table 4.2.

Tool Operation	Sample Count
Entry to Entry Search	20
Listing Operation of Internal issue entries	20
Listing Operation of External issue entries	20
Indexation Operation	18
Full Refresh	12
Database Refresh Operations	12

Table 4.2: The table show each operation's minimum sample count on each dataset.

4.12 Summary of Method

The method chapter of this thesis project is broad since it comprises many mechanisms and methods for different purposes, both regarding the implementation, but also regarding evaluation and research. To summarize this chapter to its fundamental mechanisms, there is a couple of things we can repeat and keep in mind:

- Implementation, and therefore the definition of the semantic distance, is abstracted in logic to be defined in terms of the search engine parameters set during implementation and allowed for the user to manipulate during run-time. One important parameter is the search field boosting feature.
- The boosting feature needs to be set for each unique dataset used in the tool, and therefor it is of importance that these values are set in a systematic order. We do this by examining the nature of the dataset in question, and by examining the value-variation for all entries of the dataset; a dataset that has no or small variation for a specific meta-data field should not yield high impact on the similarity score upon search as an example.
- The similarity scores are essential for the tool and its use. In order to make them understandable, they are normalized so they can be interpreted as similarity in a percentage value. The scores themselves also need to be evaluated for accuracy and reliability. We do this by setting up six datasets with special characteristics in content and size to emulate different scenarios.
- The evaluation processes of accuracy and scalability are important, and in order to perform these, quantitative methods are preferred over qualitative methods, since the data produced by quantitative methods tend to be less subjective than qualitative methods. These methods can be used with no complexities for measuring scalability. The definition of accuracy however is ambiguous and a far more abstract concept than scalability. This makes the quantitative measurement of accuracy more difficult, and therefor we solve this by also using qualitative methods.

Chapter 5

Results

In this chapter we present our results; in section 5.1 we present the semantic distance measurement metric definition we ended up with as a result of our approach method. In section 5.2 we present our qualitative as well as the quantitative evaluation for the accuracy of the tool implementation. In section 5.3 we finally present the performance of the tool, thus its scalability.

5.1 Semantic Distance Measurement Metric

The definition of the semantic distance measurement is mostly pre-defined as the search engine, Lucene, abstracts this matter away. However we can still describe the definition by the way we set the parameters and implement the tool around the search engine. The resulted measurement metric is a relative metric, a numeric score value from 0 to 100. The metric is a composition of sub scores of different meta data fields, e.g. issue title, the issue description, the project it appears in and the software component the defect impacts. The sub score for each meta data field is then weighted differently, according to deemed importance in the context of the dataset of issue artifacts.

If a dataset includes issues related to a limited amount of software components, then the software component field will have a smaller weight than it would have for a dataset with a diverse variety of software component values for its indexed issues. One can say that the resulting metric for the semantic distance is a relative numeric value that takes in consideration the uniqueness of the different artifacts in a dataset, in order to yield a higher score, in case two allegedly “unique” artifacts, are similar to each other.

5.2 Accuracy

The resulting accuracy of the tool, hence the accuracy of the metric, was decided qualitatively by observing the ranking order and score results on each set of the search setups. The results are presented in table 5.1 and table 5.2.

The accuracy for the tool was considered to be somewhat accurate since it indicated external issue duplicates with great accuracy as can seen in table 5.1 marked *External Issue Duplicate*. This consideration is a result of the manual qualitative observations of both the data but also on individual comparison of the search results.

To some extent, the metric could accurately point out the master issue entries, not only by searching on their counterpart internal issue duplicate, as shown in table 5.2, but also by searching on an external issue entry that has links to a duplicate, as shown in table 5.1, for the expected entry type of *Master Issue Entry* rows on each prepared search instance.

The metric also indicated other master entry duplicates, although it performed better when the search starting point were from the prepared search’s internal issue instead of the same internal issue’s linked external issue entry. This can be observed by comparing the *Other Duplicate* rows of the different prepared searches in table 5.2 and table 5.1.

Overall, one can also observe that the resulting score accuracies are not consistent for different sets of searches, nor did the tool always rank the search-results as expected or at least consistently. One troubling finding for a few instances is that the ranking order would be correct while the score would be unexpectedly low. This is important to point out since one could easily conclude that "this is still somewhat a good result", but in live environment cases where the correct result is not known this would indicate to the user that there is no relevant entries among the results, i.e. a false negative.

This finding also occurs in the opposite manner; Relevant and irrelevant artifacts would be in incorrect ranking order and with similarly high scores. This kind of search result implies a false positive; i.e. it would indicate that a wrongful entry is related, or more related, to the inputted artifact than the one who would be expected if the review would be done manually.

Prepared Search [Dataset].[Search №]	Expected Entry Type [Duplicate|Master]	Rank Order [Order of Entry]	Similarity Score [M-Score]
DS1.1	External Issue Duplicate	1	100
	Master Issue Entry	81	4
	Other Duplicate	23	8
	Other Duplicate	2	100
	Other Duplicate	56	5
DS1.2	External Issue Duplicate	1	100
	Master Issue Entry	8	28
	Other Duplicate	28	17
	Other Duplicate	73	17
DS2.1	External Issue Duplicate	1	100
	Master Issue Artifact	633	3
DS2.2	External Issue Duplicate	1	100
	Master Issue Entry	111	2
DS3.1	External Issue Duplicate	1	100
	Master Issue Entry	2	23
	Other Duplicate	3	22
DS3.2	External Issue Duplicate	1	100
	Master Issue Entry	16	15
	Other Duplicate	32	10
DS4.1	External Issue Duplicate	1	100
	Master Issue Entry	Entry not Part of the Dataset	
DS4.2	External Issue Duplicate	1	100
	Master Issue Entry	Entry not Part of the Dataset	
DS5.1	External Issue Duplicate	1	100
	Master Issue Entry	782	3
DS5.2	No Linked External Issue for This Prepared Search		
DS5.3	External Issue Duplicate	1	37
	Master Issue Entry	3	25
	Other Duplicates	19	7
DS6.1	External Issue Duplicate	1	100
	Master Issue Entry	Entry not Part of the Dataset	
	Other Duplicate	2	56
	Other Duplicate	3	56
DS6.2	External Issue Duplicate	1	100
	Master Issue Entry	Entry not Part of the Dataset	
DS6.3 (DS5.3)	External Issue Duplicate	1	100
	Master Issue Entry	Entry not Part of the Dataset	
	Other Duplicate	18	5

Table 5.1: Results for custom prepared searches, by searching with an external issue. All expected entries are instances of the internal issues artifact.

Prepared Search [Dataset].[Search №]	Expected Entry Type [Duplicate|Master]	Rank Order [Order of Entry]	Similarity Score [M-Score]
DS1.1	Master Issue Entry	81	4
	Other Duplicate	23	8
	Other Duplicate	2	100
	Other Duplicate	56	5
DS1.2	Master Issue Entry	8	28
	Other Duplicate	23	8
	Other Duplicate	68	17
	Other Duplicate	73	17
DS2.1	Master Issue Entry	633	3
DS2.2	Master Issue Entry	111	2
DS3.1	Master Issue Entry	2	23
	Other Duplicate	3	22
DS3.2	Master Issue Entry	16	15
	Other Duplicate	32	10
DS4.1	Master Issue Entry	Entry not Part of the Dataset	
DS4.2	Master Issue Entry	Entry not Part of the Dataset	
DS5.1	Master Issue Entry	782	2
DS5.2	Master Issue Entry	6	19
DS5.3	Master Issue Entry	2	70
	Other Duplicates	4	28
DS6.1	Master Issue Entry	Entry not Part of the Dataset	
	Other Duplicate	2	64
	Other Duplicate	3	64
DS6.2	Master Issue Entry	Entry not Part of the Dataset	
DS6.3 (DS5.3)	Master Issue Entry	Entry not Part of the Dataset	
	Other Duplicate	2	25

Table 5.2: Results for custom prepared searches, by searching with an internal issue. Note that all expected entries are instances of the internal issues artifact. Note how the results for master entry duplicates is the same regardless as in table 5.1 when the similarity score for the internal issue duplicate of the linked external issue is 100.

5.3 Performance and Scalability of Measurement

This section shows the data points measured to present the performance and illustrate the scalability of the tool in different aspects. The measurements were performed code-wise by hooking the code with execution timers. While the hooks placement could shift the measurement in different directions, the important metric is not the exact result, but the magnitude and growth of the graph these tables form.

The following subsections first present data for operations that need to involve all indexed artifacts regardless of type, such as searches, database refreshes and indexation of those. Normally these operations are of heaviest importance. The two other subsections present data where only one artifact type is involved in order to execute the listed operations; that data serves mainly the purpose of showing how the performance on one hand may shift the data depending on the artifact type, but does not impact the scalability with growing amount of data.

5.3.1 Performance for Operations Involving Both Internal and External Issues

Table 5.3 shows performance for actions that operate through all indexed entries, internal issue entries and external issue entries alike. The first column presents the total amount of entries that the listed operations need to consider and traverse through and the other columns represents different operations' execution time in milliseconds.

In figure 5.1 we can see the required execution time for each operation for different amount of indexed entries. We can conclude that the tool-specific implementations and operations scales excellently since their execution time grows by $O(log(n))$. Operations related to extraction of data from remote resources is limited by the network which the tool operates from, and do not perform as well for natural reasons. The execution complexity of those operations seems to grow by $O(n)$. By looking at the plot with the logarithmic *amount of entry* axis in figure 5.2, it becomes clearer that they grow a little faster than $O(n)$.

Total Entries [Entry Count]	Full Refresh [ms]	DB Refresh [ms]	Indexation [ms]	Entry to Entry Search [ms]
60	9 618	10	2 985	951
600	4 148	477	1 371	927
6 145	21 382	7 348	2 949	1 497
34 326	152 179	120 494	24 625	4 151

Table 5.3: Mean values of sampled performance data for global operations. We can clearly see indications of a scalable implementation, specially for the common operations such as searching and indexation.

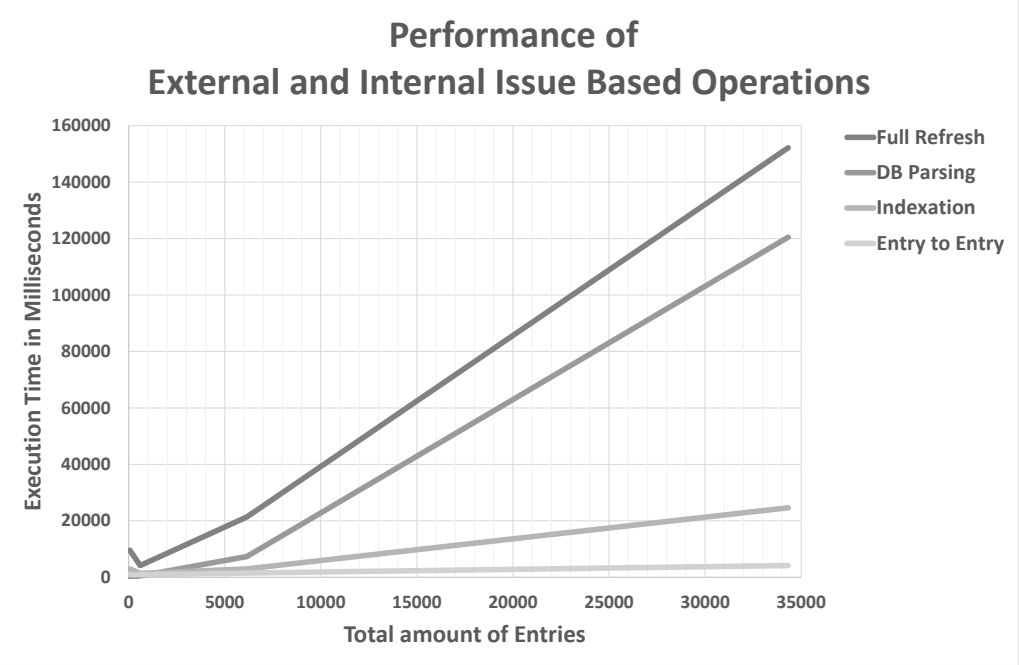


Figure 5.1: Visualization of performance data for global operations with linear axis representing entries. It can be concluded that the tool-specific implementations and operations scales excellently by $O(log(n))$, while the data-extraction operations scales well, by $O(n)$.

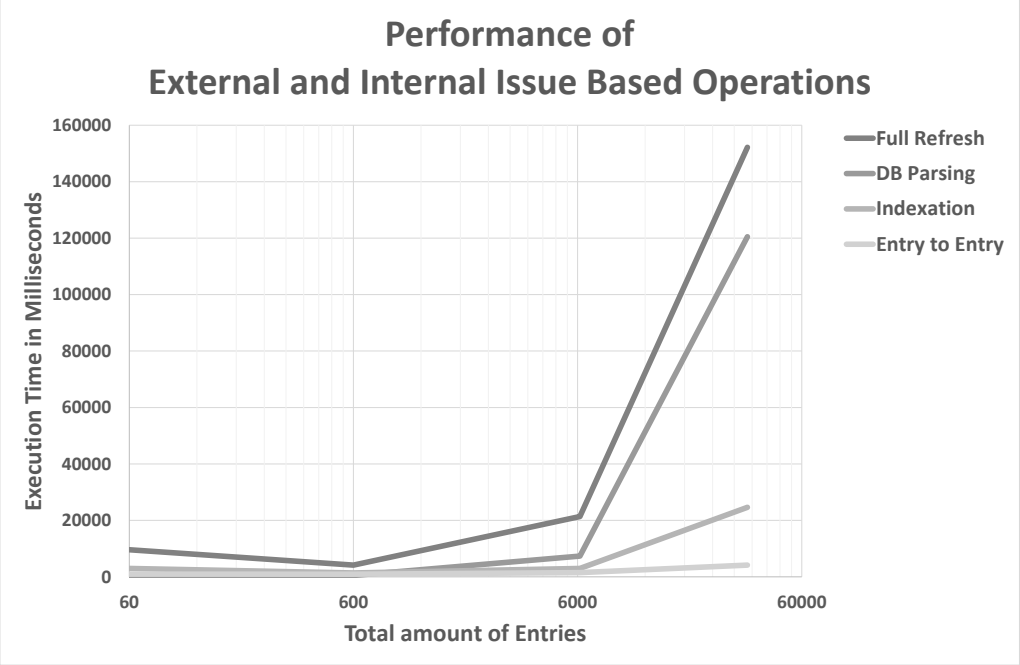


Figure 5.2: Visualization of performance data with a logarithmic axis for entries meant to show the growth of the time it takes each operation for each factor ten increase in entries. Here it is more obvious that the indexation operation actually scales in $O(nlog(n))$ and not in $O(log(n))$.

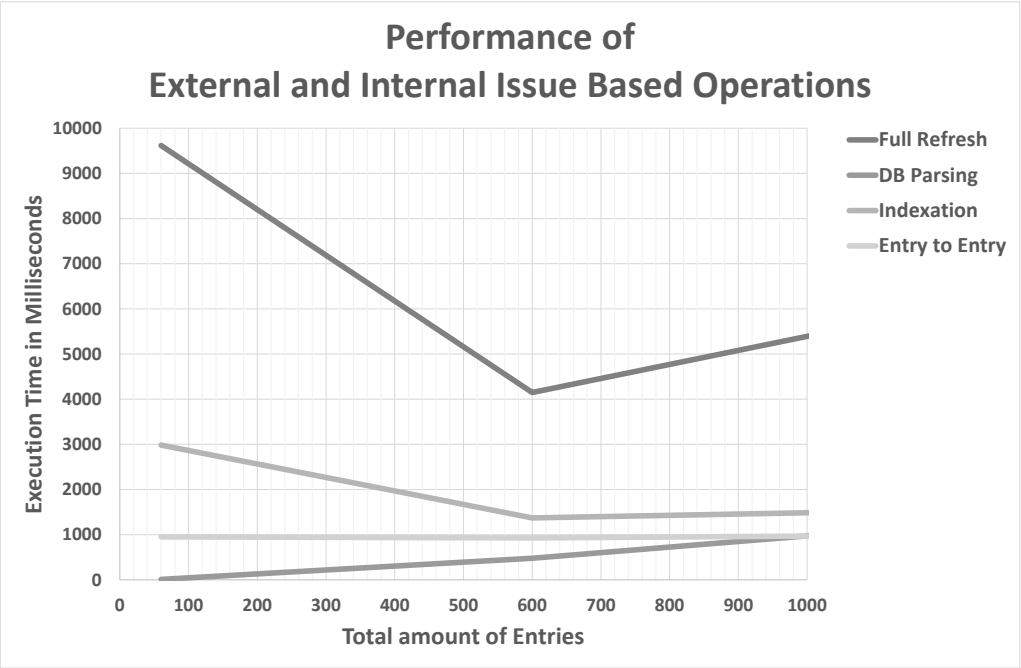


Figure 5.3: Visualization of performance data with linear axis for entries, cropped and zoomed in to the first thousand entries to give a clear overview of the initially unsteady correlation between entry-count and execution time.

5.3.2 Performance for Operations Involving Internal Issues Only

This table shows performance for actions that only operate through all indexed internal issue entries. The first column presents the total amount of the indexed internal issue entries the operation needs to traverse through, while the other column represents the operation relevant to these indexed internal issue entries and its execution time in milliseconds for the different sets of entries.

Total Internal Entries [Entry Count]	List all Internal Entries [ms]
55	605
500	630
5 466	1 010
20 000	2 484

Table 5.4: Mean values of sampled performance data for operations involving internal issue entries only

5.3.3 Performance for Operations Involving External Issues Only

Table 5.5 shows performance for operations that only operate through all indexed external issue entries. The first column in it presents the total amount of indexed external issue entries the operation needs to traverse through, while the other column represents the operation relevant to these indexed external issue entries and it's execution time in milliseconds for the different sets of entries.

Total external issue Entries [Entry Count]	List all external issue Entries [ms]
5	608
100	608
679	962
14 326	2 050

Table 5.5: Mean values of sampled performance data for operations involving external issue entries only

5.4 Summary of Results

Since the results are divided in three aspects, we can summarize them accordingly:

- The resulted semantic distance metric is basically defined in such a way that the similarity of two artifacts essentially depends on the dataset, in which they are included. The diversity among the different values for the different meta data fields has an impact on the metric.

- The resulted accuracy of the metric is ultimately acquired by measuring the results the implementation of the metric produces. There we can observe that the accuracy is not consistent through out the different scenarios. However the overall accuracy result is promising for future improvements.

- The resulted scalability of the implementation is seemingly perfect, particularly in heavy-use operations. The less scalable operations of the implementation involves data transfers over the company's network, yet they are more or less linearly scalable. Regarding the isolated, local operations; we can clearly see that the differences between the issue artifact types do not infer on the scalability, as illustrated by the parallel dotted lines in the comparative chart in figure 5.4.

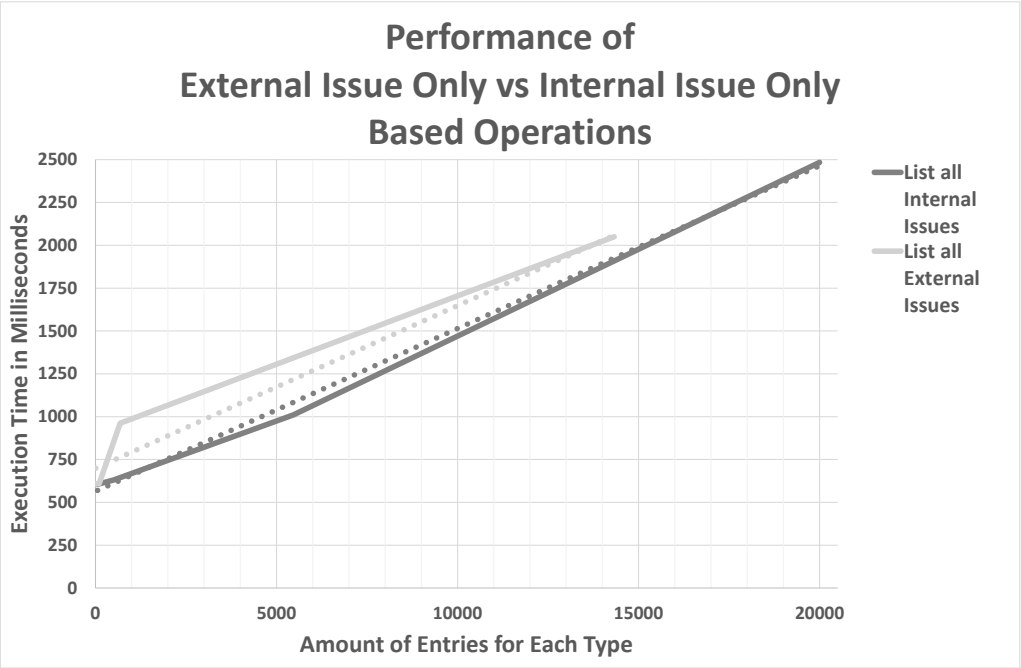


Figure 5.4: Visualization of performance data with linear scale of internal and external issue entries individually. Doted lines is supposed to illustrate their similarities in growth.

Chapter 6

Discussion

The main goals of this project, which set the basis of the research questions, were to implement an information retrieval tool based on a search engine to measure semantic distances between two semantic artifacts; examine the semantic distance assessment definition of the implemented tool; and to lastly examine the accuracy and the scalability of such a tool. This section discusses the results of this thesis work project, particularly in regard to the listed goals in chapter 1.1. We discuss what the results mean and how they answer the research questions. Different factors' impact on the results, such as error sources and project limitations, will be discussed to try to give a good insight on the results' validity. Finally we also discuss future work and how this thesis work can be developed further.

6.1 Accuracy of the Implemented Measurement of Semantic Distances

With the execution of the accuracy measurement, we seek to answer how accurate the defined assessment measurement of the semantic distances is. We do this by measuring the accuracy of the implemented tool itself to answer this research question (**RQ1**). The resulting accuracy of the tool varies between the different scenarios.

For the external issue reviewing scenario, 12 out of 13 instances of external issue duplicate searches, almost all of them, scores 100 out of 100 in the similarity score value as presented in table 5.1. The only such search instance not scoring the maximum similarity score of 100 scored 37; however it is still ranked as number 1 among the results.

For the matter of the second scenario, consisting of recovering semantic links among internal issue artifacts, the accuracy can arguably be vastly improved even if it seems promising in some cases.

It is important to discuss where and when the tool is actually accurate in one or another regard and when and where the tool seems to give random outcomes. And of course, why this is. We can observe and state that the tool achieves good accuracy when it comes to duplicates between software artifacts of different kinds, i.e. the comparison of an internal issue entry with its external issue entry counterpart, even for instances where the corresponding meta data fields between two artifacts have varied values.

In less trivial cases however, this accuracy rapidly drops. This can be seen by two indications; the expected search result scores for targeted entries and the order they rank in a search result. Studying table 5.1 for master issues as expected entry type, we can see that the rank can be considered good for 4 of 7 sampled cases, if we consider a ranking below 20 as an accurate result that compensates for the more troubling similarity scores. The results in rank and score is even more varied when looking for rankings and scores for finding other master issue duplicates as presented in table 5.1 and table 5.2.

We did manual reviews to have a good hint of what we could expect from different score values, but one can also study the table to imagine what kind of scores shall be considered good or bad. It is important to keep in mind that same scores in different datasets or even subsets of these datasets could have different indications based on the quality of documentation. In general terms a score over 56 for dataset 6 should be considered good, as the first three ranked results in the first prepared search for that dataset, *DS6.1*, shows that the highest scores is 100, 56 and 56 respectively, as shown in table 5.1. For dataset 3 this threshold is lower, at an approximate score value of 20.

6.2 Possible Approach Development to Improve Accuracy

As the defined assessment implementation is limited in different aspects, one can always improve it. There is a couple of identified factors that affects the accuracy negatively and are good to consider in order to address the at times random accuracy.

We can answer the research question regarding how the defined assessment for the semantic distance measurement and implementation of it could be improved (**RQ2**), by looking into four mention-worthy, identified factors as they make good discussion topics:

- The disregard of different meta data's nature when indexed in the search engine.
- Limited utilization of available, maybe more relevant, meta data at the Case Company.

- The primitive approach of setting and tuning meta data field boosts in the search engine.

- The accuracy measurement approach can also be improved, since flaws in the measurement could imply accuracies higher or lower than they are in reality.

These factors will be motivated and discussed in more detail in 6.2.1 through 6.2.4. It is worth pointing out that accuracy can be impacted by even more factors not mentioned here since they are outside the scope of this thesis work. However some could still be interesting as part of a future work to look at. Some of these factors are brought up in 6.4.

6.2.1 Disregard to Characteristics of Different Meta Data Fields

The Lucene search engine's algorithms is constructed for string indexation primarily, and these are used for all kinds of meta data fields of the two software artifacts to simplify the implementation of the tool.

Important numerical meta data fields of the artifacts could therefore not be used, or at least not be utilized to give better accuracy efficiency to the solution. The problem with using string indexation on numerical values is that numerical values could be similar, but indicate very different, far away things.

A hypothetical meta data field storing a simple numeral will be indexed with Lucene's string indexation method; therefore the values 300 000 and 30 of this hypothetical meta data field will yield a rather high similarity score in our solution while the values 999 999 and 1 000 000 will yield low scores in our solution.

Of course, as humans, we understand that the values in the latter example surely are far closer to each other than the values in the first example, thus the yielded similarity scores should be inverted, with a high similarity score between the numeric values of 999 999 and 1 000 000 while a low similarity score should be the result for the numeric values of 300 000 and 30.

In our solution this meant that we could not incorporate a meta data field for software version number for the tool to consider. This meta data field could be very useful since an issue on a product using an older software version should yield lower similarity scores than a more or less identical entry, but with e.g. the same, or at least closer, software version.

Submission date is a meta data field we actually chose to use rather early in the project. But since this meta data field impacted the similarity score with rather arbitrary turn-outs, we needed to considerably down-boost that field despite it's importance thus potential to contribute to better similarity scores and therefore better accuracy for the tool.

Another example of these kinds of error sources are fields that would contribute to better and more accurate scores if they would be set to give binary yes-or-no score impact. Score-wise this would change the behavior of the algorithm, so it assigns maximum weight or no weight at all on the overall score from such a field, instead of having a value based on how similar the string is.

This could be applied on different meta data fields such as the field for project name of the artifact. Whether an issue's project name has a similar project name of another issue does not tell us anything about these two different projects semantic distance to each other, and therefore should not give a higher nor lower similarity score based on this.

6.2.2 Limited Utilization of Available Meta Data

In order to limit the project scope, a limited number of meta data fields where to be examined and considered in the implementation, and after the first stage of maturity it needed to remain unaltered.

This meant that as the project proceeded there were some realizations made about the initial set of meta data fields. Some fields should have been omitted and other had better been added. As an example we came to realize that the meta data fields product and project together always formed a codename value for the platform field, making the former two obsolete and redundant. On the other hand we gradually came to understand the importance to consider meta data fields such as software version number of the issue artifact entries.

Ideally it would be preferred to include as many available meta data fields as possible to be considered and weighted by the tool's assessment measurement, making the tool better, thus also our definition of the semantic distance better. More meta data fields involved in the comparisons yields more accurate quantitative metrics, which in turn gives more accurate and consistent similarity scores. This is of course too large in scale for this project's set scope and capacity.

6.2.3 Tuning of Boosts

Boosts were configured manually for each dataset by discussing with the with the supervisor at the Case Company, a Senior Verification Architect. Trial and error were initially the main approach, but we came to establish a basic principle to go after; the variation of different possible values for each meta data field within a dataset were essential to decide the importance a meta data field should have.

We tried to automate this tuning procedure by letting the tool count each variation of the values and assign a boost value for each field based on this count. We tried to base this (semi-)automation approach by using established systematic framework called TuneR [5].

However, this whole approach was abandoned when we realized there were no real good practical implementation to come up with and put in use without the extensive and systematic research approach needed (which was out of scope for this project), particularly because of rigid and limited set of meta data fields available for the tool;

However we would still argue that this approach would have a potential progress if worked upon. We used these count values as quantitative guides in our manual qualitative decisions. While we did some low key comparisons of accuracy performance with versus without this guide, it was not in a systematic manner, and therefore we cannot state it resulted in better accuracy. However, the accuracy was not impacted negatively by this and this would provide a systematic way to set boosts.

6.2.4 Accuracy Measurement Approach

To measure the accuracy systematically and quantitatively proved to be a challenge. However it was not impossible. We measured both similarity scores and rank order of the search results in order to get a good overview of the accuracy performance. We also had a relatively good data and information to use as key source of comparison.

This was completed with the quantitative procedure of doing manual informal reviews with a Senior System Architect at the Case Company with intuitive knowledge about the handled data, to consolidate and confirm the quantitative measurement.

The choices of datasets were designed to include data of two different projects, where dataset two through four; DS2, DS3 and DS4; were subsets of dataset one, DS1 which in turn included a diverse set of data volume with artifacts describing very different issues. The very same approach where applied for the other project on datasets five and six, i.e. DS5 and DS6.

Dataset four and dataset six were designed to only include a very small amount of internal issues originating as external issue duplicates, with their linked corresponding external issues to have a better understanding on how noise would influence similarity score and ranking of expected search results for targeted entries.

The design approach for these datasets were done in in order to have different levels of difficulties to challenge the tool with and were designed in an early phase of the project in order to be able to study how the tool is affected by different volume sizes and variety of issue types. These factors proved to have smaller impact than first feared and the accuracy measurement did not have this kind of specific focus. Instead a rather more general measurement approach was employed.

The accuracy measurement could however be improved in two more ways; first we could have implemented an extensive formal review process with more senior employees as participants with a set of questions to ask them regarding their judgment of the tool accuracy.

We could also do a more overall accuracy measurement to focus on the use of the tool and how the tool actually can help, or possibly set obstacles, in their normal procedures by letting reviewing teams review a limited set of issue submissions as they usually do and with the tool to evaluate the actual added value for the company the tool could provide.

6.3 Scalability of the Tool

In response to the research question regarding the scalability possibilities of the implementation (**RQ3**) we seek to answer whether the implementation and solution approach of the tool scales with, theoretically, forever growing data volumes that needs to be indexed and searched through.

In contrary to the discussion around the research question regarding the tool's accuracy (**RQ1**), it is not suitable to use qualitative approaches to answer this questions, and luckily it is pretty straight forward to measure scalability by simply employing performance measurement and sample these on different data volumes.

The results of this measurement can be found in figure 5.1 and it is pretty clear that the tool itself is highly scalable. The operations for refreshing the database uses network communication, and even though it scales less well, it still scales linearly. The chart in figure 5.2 has a logarithmic axis for entries making it a better guide to determine how well every kind of operation scales; the database fetching operations grows by $O(n)$ while the indexation operation grows by $O(nlog(n))$ and lastly the search grows by $O(log(n))$.

The separate internal issue only and the external issue only search operations scales, as expected like the global search operation and grows by grows by $O(log(n))$ too. The chart in figure 5.4 compares the two operations operating in an external issue only and an internal issue only domain respectively, and how they grow.

One can also spot some anomalies in the graph's very beginning. The chart in figure 5.3 exposes these. Since the value for each operation is a mean value of a set of samples taken for each operation, these phenomena can not simply be dismissed as coincidence. However they can easily be explained; the search engine library methods used have probably code routines executing in constant time, causing an execution time offset large enough to make a difference on the measurement on an indexed dataset of 60 total entries, but too small to be visible on the measurement for the indexed dataset of 600 entries in total.

These constant offsets could also be explained, not only by how the search engine is implemented, but also as an effect of how we placed the measurement hooks. We did not put much effort and thought on how to place these time measurement hooks other than using general common sense.

We argue that these anomalies that could be caused by this fact does not affect the overall conclusions since the absolute measurement values and numbers are not critical nor important to answer the research question regarding scalability. Instead, relative values are important here, making the criteria of consistency through the measuring methods and the sampling procedures a critical matter; This is a criteria we fulfill. Of course, if these results were to be used to draw new conclusions where the absolute values actually mater, our data could be unsuited for the purpose.

6.4 Future Work

The tool need to be developed further to be more intuitive for the user, but also developed and improved on the issues discussed in the discussion chapter, particularly chapter 6.2.1. There could be developed new approaches to append the functionalities of the tool to possibly give the user qualitative metrics too.

Future Development of Approach:
After we have tried to implement some kind of automatic boost tuning function, we believe there could be a potential field to explore. If this could become a reality the user of the tool may no longer need to weight the fields and evaluate the weighting and re-weight them for each dataset. This would considerably increase the speed work flow for employees that need to test an artifacts on different, previously not set nor weighted, datasets.

This would also probably pave the way for implementing functionality that allows the tool to produce qualitative metrics, or even suggest different qualitative decisions and practically fully automating the whole review procedure.

Improved Qualitative Metrics:
In the first stages of this project, before it was defined, we envisioned a tool that would compare two artifacts and give a suggestion on whether that pair of artifacts were related and should be linked or not, as a qualitative assessment instead of a quantitative metric to aid a human. This proved to be difficult to implement. However we still believe this would be possible to achieve.

If this is achieved and the development is extended in this field the possibilities of automation can be endless; imagine an AI that by itself does comparisons between artifacts, unattended, and finds possible related artifacts, artifacts with abnormal data and duplicates.

The Human Factor and the Graphical User Interface:
An important consideration is also that the human users of the tool have a great effect on its accuracy since a wrongful use of it, e.g by not understanding the tool or its function, will have an overall negative effect on the conclusion and decision. The human factor is also to be considered *before* the use of the tool, namely when artifact entries is submit-

ted to the issue database and reviewed. Incomplete data, and inconsistently formulated issues by many different people plays a negative role in achieving higher accuracy. Sometimes the data could be straight out wrongful because of misunderstandings or by simple carelessness.

Another factor has also to do with the design of tool towards the user of it. The tool provide the field-to-field similarity score presentation to help the user understand why a certain pair of artifacts achieved a certain similarity score by the tool.

This is intended to primarily help the user user to devalue or revalue the overall score if the tool seems to prioritize wrong traits and boost irrelevant fields. It could also help the user understand where a boost modification can be necessary to yield accurate similarity score. Normalization of these similarity score values were however necessary to apply in order to have an understandable way to interpret them.

We think our current normalization approach is best suited to help the user of the tool understand how the similarity between two given artifacts matches based on an unit of share, say percentage unit, to understand how much each meta data field contributed to the yielded overall similarity score.

However it could give the wrong impression in some cases: if we have a case where the overall similarity score between two artifacts yielded a very low similarity score e.g. a score of 3, and the only field actually impacting this score would be e.g. the title, the field-to-field similarity score would be 100. In this case it is probably obvious what the score implies, but in more complicated situations this could be interpreted as the titles between the two artifacts "match 100 percent" thus potentially resulting in a misinformed and potentially wrongful decision based of this interpretation.

An alternative normalization approach could be to normalize the field to field similarity score around a relative distribution value instead of an absolute one, as in the current case. Practically this would mean that the share would simply be multiplied with the overall score, thus getting a relative distribution value where the sum of all field to field similarity scores would equal the overall similarity score instead of 100. This approach would not affect the tools scale up performance.

Within the frames of this thesis work's implementation we only included the listed elements of user aid to help the user understand the tool. Small efforts were put in designing this tool and frankly not much has changed from its previous life as the Case Digger; it even had a fairly different purpose as a change impact analysis tool [4].

The usefulness of software tool need as good user interfaces as they need good implementation models. Future works based on this work could do everything from actually evaluating tool's user interface to actually redesign the user interface with the user experience perspective in focus.

Regretfully these small efforts put on e.g. designing a better graphical user interface was a result of project scope trade off; also no evaluation of the graphical user interface's effects and how it is perceived by the targeted users were performed whatsoever for the same reason. In a future wok maybe an evaluation can be made on the current tool and on an designed version to compare and find out if the tool can be helpful if altered in design.

Chapter 7

Conclusions

The aim with this thesis is to investigate if it is possible to have a quantitative measurement on this difference by using a search engine as our information retrieval technology core. We did define three research question to answer this, but we also defined two common scenarios for when the implementation of the tool can be useful, since this is relevant for the practical part of the thesis work.

The work implementation's main problem where described in two scenarios: reviewing external issue entries and recovering semantic links between internal issue entries we can begin by drawing conclusions from there. The conclusions to these scenarios also answers the research question considering the semantic distance measurement definition's accuracy and usefulness (**RQ1**), and are directly derived from the obtained results.

Given the results and the arguments in the discussion for the first scenario, the accuracy of finding external issue duplicates, we conclude that the tool is highly accurate and therefore useful for for the first scenario for reviewing external issue entries to recover possible semantic links to existent internal issues. As for the second scenario, the accuracy of finding semantic links between internal issues, the accuracy outcome is rather mixed. We conclude that the tool in its current implementation is not dependable for the second scenario, and therefor it is not wise to put all dependency of the process on the tool when used in a live environment for the same scenario. We also conclude that the results show it has great potential of providing benefits if the approach is refined and further improved upon.

Regarding the research question of how this semantic distance assessment definition and the tool implementing it, can be developed and refined to improve the accuracy and efficiency of it (**RQ2**), we made conclusions during the thesis work process rather than from results. We conclude that there are four main factors to refine in order to improve the accuracy and usefulness of the semantic distance assessment definition and the tool implementing it; make the search engine consider the characteristics of the meta data subject to indexation, broaden the use of the available meta data to utilize more relevant meta data and a sophisticated variation of them, develop further an systematic and assisted method to tune field boosts and last but not least, evaluate the accuracy measurement approach and improve it.

Given the results of the performance tests we conclude the tool is highly scalable, thus answering the research question on the same issue (**RQ3**). The only threat to that conclusion in theory is the issue of network communication speeds for very large amount of data. But even that considered, there should be no problem in practice, and if that would be the case, we know that the tool and the semantic distance assessment definition itself is not what affects the bottleneck; hardware does.

To sum up; we conclude the tool could be helpful if used in limited deployment at the Case Company hosting its implementation. We can also conclude that this approach has huge improvement potential to further increase accuracy, thus extending the use cases of the tool and could ensure semi automated review and clean-up processes. However, in its current state it could be confusing to use given the fact that it will need a learning curve to use it properly and that small efforts were put to design it to be user friendly. This effort balanced up with its current return value might be considered not worth the cost just yet.

Bibliography

[1] Alex Dekhtyar Ashlee Holbrook, Jane Huffman Hayes. Toward automating requirements satisfaction assessment. In *17th IEEE International Requirements Engineering Conference*, pages 149–158, 2009.

[2] Elizabeth Bjarnason and Helen Sharp. The role of distances in requirements communication: A case study. 2015.

[3] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Information and Software Technology*, 70:204 – 219, 2016.

[4] Markus Borg. *From Bugs to Decision Support – Leveraging Historical Issue Reports in Software Evolution*. PhD thesis, Lund University, 2015.

[5] Markus Borg. Tuner: a framework for tuning software engineering tools with hands-on instructions in r. *Journal of Software: Evolution and Process*, 28(6):427–459, 2016.

[6] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[7] Jean-Pierre Corriveau. Traceability process for large oo projects. *Computer*, 29(9):63–68, Sep 1996.

[8] Andrea De Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk. *Information Retrieval Methods for Automated Traceability Recovery*, pages 71–98. Springer London, London, 2012.

[9] Bogdan Dit, Meghan Revelle, and Denys Poshyvanyk. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Software Engineering*, 18(2):277–309, 2013.

[10] Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Commun. ACM*, 41(12):54–62, December 1998.

[11] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. A comprehensive characterization of nlp techniques for identifying equivalent requirements. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 18:1–18:10, New York, NY, USA, 2010. ACM.

[12] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, Apr 1994.

[13] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Kumar Sundaram. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, Jan 2006.

[14] Matthias Heindl and Stefan Biffl. A case study on value-based requirements tracing. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 60–69, New York, NY, USA, 2005. ACM.

[15] Carl K. Chang Jane Cleland-Huang and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, Sept 2003.

[16] Stephen R. Schach Liguo Yu and Kai Chen. Measuring the maintainability of open-source software. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 7 pp.–, Nov 2005.

[17] Björn Regnell Markus Borg, Krzysztof Wnuk and Per Runeson. Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[18] Thomas Olsson. Software information management in requirements and test documentation. *Licentiate thesis, Lund University*, 2002.

[19] Austen Rainer Per Runeson, Martin Host and Bjorn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Publishing, 1st edition, 2012.

[20] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*, pages 499–510, 2007.

[21] George Spanoudakis and Andrea Zisman. Software traceability: A roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, 2004.

[22] Jennifer Whyte, Angelos Stasis, and Carmel Lindkvist. Managing change in the delivery of complex projects: Configuration management, asset information and 'big data'. *International Journal of Project Management*, 34(2):339 – 351, 2016.

EXAMENSARBETE Measuring Semantic Distances between Software Artifacts to Consolidate Issues from the Development and the Field
STUDENT Mahmoud Nasser
HANDLEDARE Elizabeth Bjarnason (LTH), Markus Borg (SICS)
EXAMINATOR Per Runeson (LTH)

Mätning av semantiska avstånd mellan mjukvaruartefakter med användandet av ett sökmotorbaserat verktyg

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mahmoud Nasser**

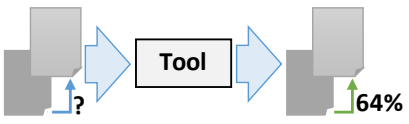
Regelbundna manuella granskningar för att identifiera länkar mellan ständingt tillkommande mjukvaruartefakter är viktigt för att hålla ordning och reducera kostnaden för ett mjukvaruprojekt. Tidskrävande är det också; därför utreder detta arbete hur ett verktyg kan åtgärda detta med en snabbare och effektivare granskningsprocess.

Olika mjukvaruartefakter, dokument som beskriver mjukvarufunktioner, ser ibland väldigt olika ut även om de behandlar samma del i mjukvaran. En funktion måste nämligen beskrivas på olika sätt för olika sammanhang; vanlig naturlig text för att kund och utvecklare ska vara överens om funktionsutformning, programkod eller testfall i tekniska respektive testningssammanhang.

Olika artefakter som beskriver en funktion har en länk mellan sig eftersom de är lika i funktion och hör ihop. Ju mindre lika två artefakter är i detta avseende, desto större *semantiskt avstånd* de sägs ha. Dessa länkar kan förloras och förändras allteftersom mjukvaran ändrar skepnad. Om man inte underhåller denna data så blir det problem senare när man t.ex. upptäcker en bugg, då man i sammanhang av programkod eller testfall sällan förstår vad som går fel utan att hitta relevanta artefakter.

Detta brukar hanteras genom regelbundna granskningsmöten där artefakter som förändrats gås igenom och man lägger till, tar bort och länkar om mellan olika artefaktpar utifrån frågan om de beskriver samma funktion. Granskningarna, som utförs manuellt, är tidskrävande och kostsamma.

Vi har med vårt arbete utvecklat ett verktyg som använder en sökmotor för att indexera två olika men besläktade defektartefakttyper. Vi definierar ett sorts mått för semantiska avstånd och matar in en defektartefakt i verktyget för att få förslag på alla andra artefakter som är semantiskt nära. Verktyget analyserar olika sorters länkar och defektdata för att skapa ett helhetsmått. Träffsäkerheten evalueras genom att testa verktyget med förberedda sökningar som facit.



Verktyget sätter ett mått på semantiska avstånd.

Våra resultat visar att för vissa syften blir verktyget väldigt träffsäkert, medan det i bästa fall blir måttlig träffsäkerhet för flera andra syften. Vi har kommit fram till att detta osäkra och blandade resultat beror på den begränsade defintionen i vad för sorts defektdata man tar hänsyn till. Oavsett detta, snabbar verktyget upp granskningen, och i bästa fall, särskilt ifall det utvecklas vidare, kan det hitta fler komplexa och användbara länkar.