Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Telecommunication Engineering Graduate Program

Semi Automated Acceptance Testing Using Information Retrieval

By
Debela Tadesse

Advisor

Surafel Lemma (PhD)

A Thesis Submitted to School of Electrical and Computer Engineering, in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Telecommunications Engineering

November, 2021
Addis Ababa, Ethiopia

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Telecommunication Engineering Graduate Program

Semi Automated Acceptance Testing Using Information Retrieval

By: Debela Tadesse

Signed by the Examining Committee:

Surafel Lemma (PhD)	_____	_____
Advisor	Signature	Date
Fitsum Assamnew (PhD)	_____	_____
Examiner	Signature	Date
Mesfin Kifle (PhD)	_____	_____
Examiner	Signature	Date
_____	_____	_____
Chair or School Dean	Signature	Date

Declaration

I, the undersigned, declare that this thesis is my original work, has not been presented for a degree in this or any other university, and all sources of materials used for the thesis have been fully acknowledged.

Name: - Debela Tadesse Signature: _____

Place: Addis Ababa

Date of Submission: _____

This thesis has been submitted for examination with my approval as a university advisor.

Surafel Lemma (PhD) Signature: _____

Abstract

Requirements Engineering(RE) is the discipline of eliciting, analyzing, specifying, validating and managing needs and constraints of a software product. It attempts to document the ideas and needs of the product's stakeholders in the software requirements specification (SRS) document. Later, when the product is developed and delivered, stakeholders perform acceptance testing. Acceptance Testing is the verification conducted to use it that a software product provides expected behaviors, as expressed in requirements. It is a testing technique performed to determine whether or not the developed software has met the stakeholder's requirements specified initially. One of the first tasks in acceptance testing is to know where a particular requirement is implemented in the software system. As the number of requirements increase, identifying where a particular requirement is implemented in before system becomes challenging and time-consuming, especially in environments where there are not so many expertise.

This thesis proposes an information retrieval (IR) based approach to address this problem. The approach uses the use cases in the SRS and the feature descriptions in the software manual, as a query and corpus, respectively, to identify where the use cases described in the SRS are implemented in the software system.

To evaluate the proposed approach, we conducted an experiment using two datasets collected from two software products. We observe that the recall values are 100% for all queries. This means that our approach efficiently associates uses cases with their implementation of features in the corresponding software. The average value of F-measure is 44% and 67% in dataset one and two respectively. This indicates that, the proposed approach is able to associates uses cases and features implementation. The Reciprocal rank values of all queries is 1, meaning that the most relevant feature are found at first rank.

Hence, the software testers would easily identify the features to test a software product.

Keywords: *Acceptance Testing, Information Retrieval, Software Requirement specification, software product, Query, Corpus, document, metric, Lucene, Indexing, searching.*

Acknowledgment

First and foremost, I would like to thank the almighty GOD for giving me healthy and keep me in all circumstances.

Second, I would like to express my deep and sincere gratitude to my advisor Dr. Surafel Lemma for his excellent support and guidance from the beginning of the thesis. Since the beginning of the thesis study, his patience and tolerance with his immense knowledge motivated me to have more interest in studying the area. Thank you for giving your time to explain all those things I should have known already. Thank you very much.

I would also like to thank my examiners, Dr. Fitsum Assamnew and Dr. Mesfin Kifle for all the constructive comments given to me during the thesis proposal and progress report presentation.

Thanks must go to my company, Ethiotelecom for giving me the chance to the program. Also, to the requirement analysis team at Ethiotelecom who helped me in providing valuable information during data collection.

Finally, I would like to thank my wife, Sinkinesh Mekonnen for her contributions to my life, constant support, and patience from the beginning to the end of the program.

Table of Contents

List of Figures

List of tables

List of Acronyms

BM25 Best Match 25

CSM Cosine Similarity Measure

DCM Dice Coefficient Measure

FD Feature Description

IR Information Retrieval

JSM Jaccard Similarity Measure

LSI Latent Semantic Indexing

MBT Model-based Testing

PM Probabilistic Model

RR Reciprocal Rank

RE Requirements Engineering

RT Requirement Traceability

RSL Requirements Specification Language

SRS Software Requirement Specification

SVD Singular Value Decomposition

SUT System Under Test

TF-IDF Term Frequency-Inverse Document Frequency

UC Use Case

VSM Vector Space Model

CHAPTER ONE: INTRODUCTION

Requirements Engineering(RE) is the discipline of eliciting, analyzing, specifying, validating and managing needs and constraints on a software product. It attempts to document the ideas and needs of the product's stakeholders in the software requirements specification (SRS) document. Successful RE requires a very good understanding of the business domain, the environment in which the system will be running, and the needs of the project's stakeholders such as customers and users [3]. Later, when the product is developed and delivered, stakeholders perform acceptance testing. Acceptance testing is the verification that a software product provides expected behaviors, as expressed in requirements. It is a testing technique performed to determine whether or not the software system has met the requirement specifications [1]. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it has met the required criteria for delivery to end-users. Acceptance testing is one of the final stages of testing, where the product is presented to the customer to validate if all the requirements are met, and the acceptance criteria for the product are satisfied [20].

Currently, many researchers proposed different approaches and tools to automates acceptance testing. Concordion and FitNesse are open source tools used to automate acceptance testing for Java based software systems. The tools setup seems to be working well for small and medium size projects and the approach can be extrapolated to larger projects. The use of Concordion assumes that the requirements are written in native speaking language and kept in a set of HTML files [27]. Fit (Framework for Integrated Test) is an open source framework used to express acceptance test cases and a tool for improving the communication between analysts and developers. It lets analysts write acceptance tests in the form of tables (Fit tables) using simple HTML or even spreadsheets. A Fit table specifies the inputs and expected outputs for the test [29].

Cucumber or Robot Test Framework tools are constrained in that the user is forced to use the keywords pre-specified in their framework to create higher-level keywords. However, other research [7] uses natural language processing techniques to reduce this overhead by not creating excessive higher-level keywords and instead allowing free form test cases to automated the

acceptance testing. These tools require an extensive knowledge of the scripting language to create functional tests. Model-based Testing (MBT) technique is another approach that generates test cases from abstract representations of the system, named models, either graphical or textual. This approach uses RSL (Requirements Specification Language), where each requirement is aligned with RSL Test Cases specifications and generate Test scripts that can be executed automatically by the Robot test automation tool over the System Under Test (SUT) [20].

The existing acceptance testing approaches usually target expert software testers. In Ethiotelecom, the requirement analysis section is preparing the SRS to develop or adopt software projects. To accept the developed software product, they use manual method of acceptance testing. This acceptance testing is done by using SRS document and manually cross-checking to verify and validate the implementation of software requirements. As the complexity of software increases, using manual method for acceptance testing becomes too challenging and time-consuming. Hence, the automated acceptance testing approach that supports non-expert requirement analyst to test requirement implementation is needed.

1.1 Statement of the Problem

In Ethiotelecom, the software development options are outsourcing and in-house development based on drafted SRS documents. Software systems are constantly evolving becoming more complex, as the number of requirements increased.

Currently, many companies adopt automated acceptance testing tools to check requirements implementation. These tools require experts. However, in Ethiotelecom, there are very few experts and limited resources. Hence, the experts manually conduct acceptance testing to see if the requirements specified in the SRS document are implemented. The manual acceptance testing requires to first identify the feature that implements a requirement and then execute the feature to see if the feature is implemented as described in the SRS. Manually looking for a feature that implements a requirement is time consuming, especially for large software systems.

In this research, we investigate the approach that would help non-experts and non-experienced software testers to identify features that implement a given requirement to automates the acceptance testing. In particular, we study and investigate the application of Information Retrieval (IR) based approach with SRS and software user manuals to verify if requirements listed in the SRS are implemented in the delivered software system.

1.2 Objectives

1.2.1 General objective

This thesis aims to solve one of the challenges of acceptance testing phase, locating the features that implement a use case description in SRS document.

1.2.2 Specific objectives

The specific objectives to be achieved in this thesis are as follows:

- To develop IR based approach that identifies features implementing a requirement.
- To evaluate the proposed IR based approach.

1.3 Scope of the Study

The scope of this thesis is focused on automating one of the activities of acceptance testing: locating a feature that implements a requirement. It will not check if the implementation works as it is described in the requirement.

1.4 Research Methodology

The research methodology followed in this thesis has the following phases.

Research problem: In this phase, the statement of the problem is identified based on background the limitation of existing approaches.

Data collection: Based on specified problem we collect two datasets to conduct an experiment.

Literature review: Under this section, we will review the state of art related to the identified problem.

Data analysis: In this phase, preprocessing is done on the datasets use for experiment.

Experiment: We will conduct experiment using preprocessed datasets. The experiment aims to see the applicability of IR based approach in acceptance testing.

Result and Discussion: In this phase, using the generated result, the result of the experiment will be discussed and analyzed.

Figure 1.1 shows the research methodology process followed in this thesis.

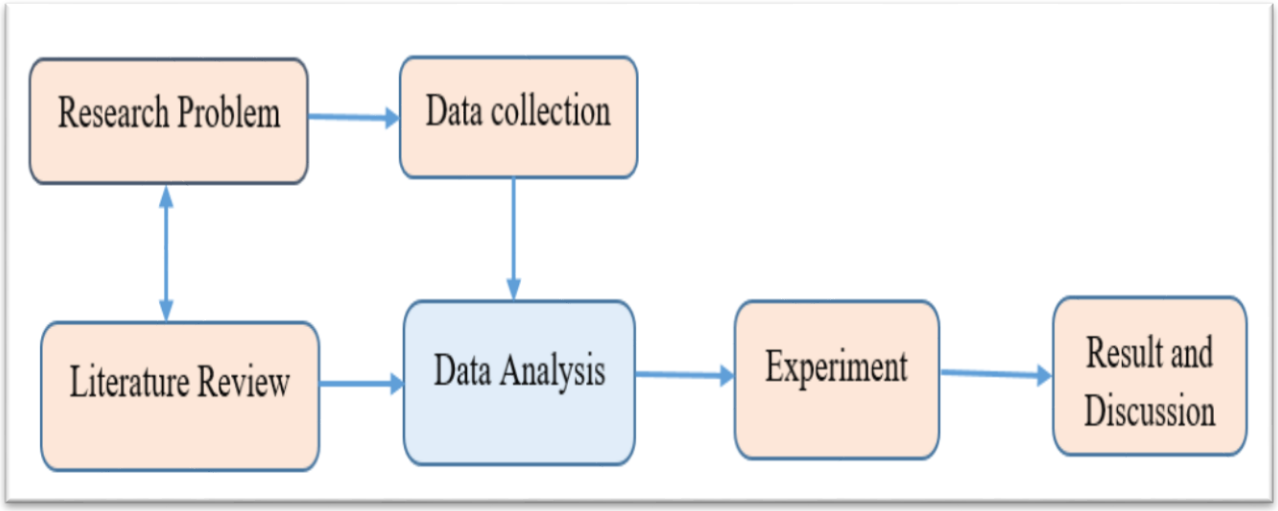


Figure 1.1: Design of Research Methodology

1.5 Contribution of the Thesis

This thesis proposes IR-based approach to assist non-expert software testers in identifying features that implement a requirement. The approach will reduce testing time and enable impales with few resources to easily test the delivered software system.

1.6 Organization of the Thesis

This thesis consists of six chapters and contains different sections under each chapter. Those are as follow:

Chapter One: It presents the motivation of the thesis and the research problem along with the objectives and contributions.

Chapter Two: It provides background information on acceptance testing, information retrieval and discusses the related works.

The proposed approach is detailed in Chapter Three.

Chapter Four: Presents the experimental design along with the description of the dataset used in the experiment and the results and discussions.

Chapter Five: Concludes the thesis and gives future directions.

CHAPTER TWO: LITERATURE REVIEW

This chapter presents an overview of related works found in the area of acceptance testing and IR approach. It is divided into three sections. The first section provides a brief description of acceptance testing, approaches used in acceptance testing and acceptance testing automation. The next section presents the basics of information retrieval approach, while the last section discusses the related works.

2.1 Acceptance Testing

Acceptance tests are defined as “customer tests,”, “customer-inspired tests,” and “conditions of satisfaction”. In systems and software engineering it is defined as “formal testing conducted to enable a user, customer, or other authorised entity to determine whether to accept a system or component.” It is also used to evaluate whether the customer requirements are met. An acceptance test is carried out by users who take the initiative in confirming whether the software under development implements the requirements of the users [30].

Software systems are constantly evolving becoming more complex, which increases the need for efficient and regular testing activity to ensure quality and increase product confidence. Software systems’ quality is usually evaluated by the software product’s ability to meet the implicit and explicit customer needs. For this purpose, customers and developers must achieve a mutual understanding of the features of the software that will be developed [20].

Acceptance Testing is an indication both to developers and to product owners that the software product satisfies the requirements specification. Requirements continue to be traced through user acceptance testing, ultimately demonstrating that the delivered solution conforms to the defined requirements and meets the needs of the client. The success of any software product lies with the fact that the customers/users accept it. It provides confidence that the delivered system meets the business requirements of the customer (and users) [27].

The main purpose of acceptance testing is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end-users. It gives the customer confidence that the system under development has the required features and behaves correctly. It gives customers the chance to check if everything matches their expectations and if the requirements have been communicated and implemented correctly. It is carried out by users who take the initiative in confirming whether the software under development implements the requirements of the users [19].

Acceptance tests have a greater relationship with the requirements. They are used to test concerning the user needs, requirements and business processes, and are conducted to determine whether or not a system satisfies the acceptance criteria and to enable users, customers, or other authorized entities to determine whether or not shall accept the system. Acceptance testing is a validation activity, performed by the customer just before the system is delivered and aimed at judging if the product is acceptable. It can be used as a mechanism to define acceptable external quality of software product [20].

One of the main reasons for the failure of many software projects is a mismatch between the customers' expectations and the pieces of requirement implemented in the delivered software product. Performing acceptance testing manually is usually challenging, expensive and time consuming. Hence, automating acceptance testing is a promising initiative to improve and ease the testing process. However, the Automation of acceptance tests may thus seem like a promising initiative to ease and improve this process [29].

Acceptance testing can be used to track and evaluate the software development process by measuring the number of tests that pass or fail. It is used for comparing a system to its initial requirements and the current needs of its end-users and for verifying whether the system implements the requirements as intended. One of the main reasons for the failure of many software projects is the late discovery of a mismatch between the customers' expectations and the pieces of functionality implemented in the delivered system [21].

Automating Acceptance Testing is closely tied to requirements specifications and provides a mechanism for continuous validation of software requirements. The automated acceptance tests are derived from the customer requirements. The focus of these tests is to verify the completeness of the requirements' implementation. Acceptance tests are used to determine if a system performs according to the contractual requirements [31].

Automated acceptance relies on various information retrieval algorithms that use techniques such as the vector-space model or the probabilistic network model to compute the likelihood of a link based on the occurrence and distribution of terms. It also uses scripts and tools that prepare data and a state, then executes the steps required to verify the scenario in an automated way [22].

This approach helps end-users to validate and verify the behavior of the final state of the product. The automated acceptance testing is more reliable and quicker than manual testing. In addition to reducing manual effort, time and resources dedicated to the development of tests, it ensures a higher quality of requirements implementation. It allows reducing the time and resources spent with them [20].

The primary purpose of acceptance testing is verifying and accepting the behavior of the final product, which means the software is being developed. The use of automated acceptance testing tools ties acceptance tests into the requirements specification, which results in better maintainability. The successful run of acceptance tests is an indication both to developers and to product owners that the software product satisfies the requirements specification [27].

2.2 Information Retrieval

Information Retrieval (IR) is the process of discovering documents relevant to an information request. The main aim of IR is to identify relevant documents in document collections given user-specified information needs. In practice, the query and the searchable documents could be any type of software artifact in the system. Information Retrieval methods have been largely adopted to identify traceability links based on the textual similarity of software artifacts [14].

The IR approaches are generally based on chunks of natural language text. It returns a set of matching documents from a set of documents (corpus) for a given query. First, extracting key terms from each document in the corpus and then computing the similarity between the terms of the query and the key terms of each document. A list of candidates is then presented to the user who has to decide which of the candidates are relevant to his query [4].

Most IR methods work by converting each document in the collection into a mathematical representation to capture the information content of the document, after that a comparison is conducted with similar representations of user information needs (queries). IR attempts to model individual documents within document collections and to model user information needs. IR methods determine how relevant the document representation is to the query that represented the user query [10].

2.2.1 Information Retrieval Techniques

Information Retrieval techniques are used to link artifacts to each other via a mechanism that queries a set of artifacts. By using an indexing process based on pre-processing documents, identifying attributes, or developing a thesaurus of keywords, artifact information and semantics are parsed and used to rank the artifact on its relevance to a given query. IR-based methods recover traceability links based on the similarity between the text contained in the software artifacts [14].

The IR techniques mostly used to identify links in documents are Latent Semantic Indexing (LSI), Probabilistic Modelling (PM), and Vector Space Model (VSM). Below we briefly describe each technique [8].

1. *Vector Space Model (VSM)*

In this method, the document and the query are represented in the form of vectors of keyword weights. The keywords, or index terms, are words that occur in the document collection. Documents are then ranked against queries by computing distance functions between the corresponding vectors. It represents both documents and queries as vectors in a high-dimensional space and similarities are calculated between vectors using distance functions [12].

VSM treats documents and queries as vectors; documents are ranked against queries by computing a distance function between the corresponding vectors. Let D = {d_1, d_2, d_3,…, d_N} be the set of N documents that represents the corpus and Q is the queries that are represented in the same way as documents. The similarity between a document d_i and query document q_j in Q is computed as the cosine of the angle between the corresponding vectors [11]. The vector space model measures the similarity between the document d_i and the query q_j as the correlation between the two vectors d and q [18].

2. *Latent Semantic Indexing (LSI)*

LSI is an approach to reduce the dimension space, sometimes successful in reducing the effects of synonymy and polysemy. It is based on concept matching with the purpose to improve simple word matching. It applies the Singular Value Decomposition (SVD) technique to map terms and query into a lower-dimensional space associated with concepts. To do so, software artifacts are regarded as textual documents. Occurrences of terms are extracted from the documents to calculate similarities between them and then classify together similar documents [12].

3. *Probabilistic Network (PN) Model*

The documents (artifacts) are ranked according to the probability of being relevant to a query. It uses a flexible mathematical framework to model queries and documents as variables in a concept space defined by the keywords in the *documents collection* (a formed as the union of all terms found in all documents). In each model, one type of particular artifact treats as a query and another type of artifact treats as a document being searched in terms of the query. PN computes the ranking scores as the probability that a document D_i is related to the component Query Q [13].

4. *TF-IDF*

Term Frequency-Inverse Document Frequency (TF-IDF) is generally a content descriptive mechanism for the documents. The TF is the number of times a term appears in a document and is calculated as follows: t_f = (Number of occurrences of a keyword in that particular document) / (Total number of keywords in the document). IDF measures the rarity of a term in the whole

corpus. The concepts of TF and IDF are combined, to produce a composite weight for each term in each document. TF-IDF is calculated as t_f-$id_f = t_f * id_f$ [12].

2.2.2 Information Retrieval Systems

Information retrieval systems compute for each document in a collection, a score that estimates the similarity between that document and a query. In typical systems, each score represents an estimated probability that the document is relevant to the information need expressed by the query. Once the process of scoring is complete, documents are presented to the user in decreasing score order, in the expectation that the user considers them in sequence until their information need has been satisfied. The three main elements of IR system are; a document collection (corpus), a set of information needs (queries), and relevance judgments telling what documents are relevant to these information needs [17].

A document-based IR system typically consists of the following subsystems [26]:

Document Representation: - is the how to select proper index terms and the representation is proceeds by extracting keywords that are considered as content identifiers and organizing them into a given format.

Query: - It transforms the user's information need into a form that correctly represents the user's underlying information requirement and is suitable for the matching process.

Matching Algorithms: - are the matching functions used to matching queries and documents. The common type of matching algorithm are Bag of Words Based and Word embedding based.

User: - is a person who put the request/information needs on the information retrieval system.

Relevance judgment: - Documents retrieved by IR systems are arranged according to their relevance to a given search query. A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need.

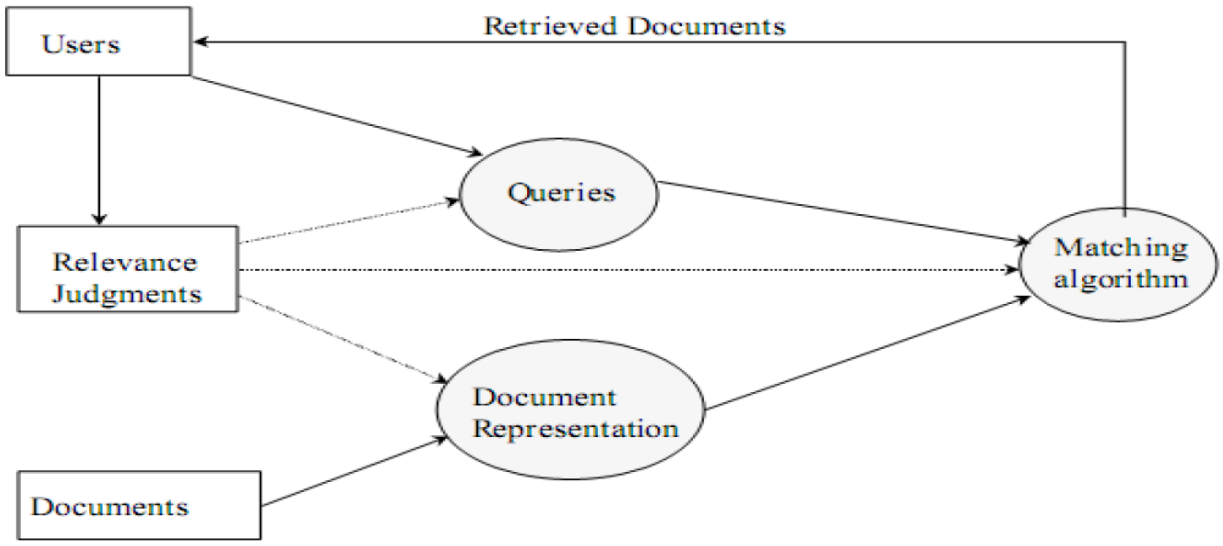


Figure 2.1: Basic architecture of IR system [26]

2.2.3 Measuring the Effectiveness in IR

IR system retrieval performance can be evaluated by calculating two metrics: the percentage of actual matches that are found (recall) and the percentage of correct matches as a ratio to the total number of candidate links returned (precision). Recall and precision are appropriate characteristics of the quality of candidate lists. In the context of a ranked list of documents, recall and precision can be measured at each document retrieved, at each relevant document retrieved, or at recall percentiles in the ranking [10].

Recall: is the ratio of the number of relevant retrieved over the total number of relevant in the collection. Recall is used to measure the ability of the search to find all of the relevant items in the corpus. It is a coverage measure. It is the measurement of the ability to retrieve correct results. the number of relevant documents retrieved R, related to a total number of relevant documents in the collection C, i.e., R/C. High recall indicates that most of the relevant items have been retrieved [9].

Precision: is the percentage of the matches within the recall set that are relevant to the query. The relationship between the number of retrieved relevant documents R, concerning a query statement Q and the number of documents D that have been retrieved based on it, i.e., R/D. High precision

indicates that most of the retrieved items are relevant. It is the measurement of the accuracy of retrieved results [6].

F-score or F-measure: is the weighted harmonic mean of precision and recall. It is computed as:

$$F = \frac{2(P * R)}{P + R}$$ where P= Precision and R= Recall

A high F-measure is achieved only when both recall and precision are high. Determination of the maximal F can be interpreted as an attempt to find the best possible compromise between recall and precision.

Reciprocal Rank (RR): This is a common measure of performance evaluation in information retrieval that calculates the first relevant document is retrieved. It is 1 if a relevant document was retrieved at rank 1, and it is 0.5 if a relevant document was retrieved at rank 2 and so on.
Reciprocal rank is computed as:

$$\boldsymbol{RR = \frac{1}{K}}$$, where K= rank position of first relevant doc.

2.2.4 Term extraction in IR methods

How the key terms are extracted from an artifact depends on the type and format of the artifact. If the artifact is a text in natural language, keywords can be extracted using ordinary information retrieval approaches by first eliminating all stop words and then stemming the remaining words. If the artifacts are models or codes, identifiers, names, and other attributes can be used as keywords [4].

2.2.5 Similarity Metrics

A similarity measure is a function, which determines the degree of similarity between a pair of textual objects. IR-based methods recover traceability links based on the similarity between the text contained in the software artifacts. Given a document vector d and a similarly computed query vector q, the similarity between them is computed as the cosine of the angle between the two vectors. Before the similarity measure can be calculated the words of each sentence have to be extracted [11].

The higher the textual similarity between two artifacts, the higher the likelihood that a link exists between them. The similarity between artifacts is computed by calculating the distance of the corresponding interpolation curves. Based on the term-by-document matrix representation, different IR methods can be used to rank pairs of source and target artifacts based on their similarities [15]. Each document is ranked according to its degree of similarity score with the query q. A retrieval strategy is defined by specifying a threshold level for the similarity index. Therefore, documents with a similarity coefficient above that level will be retrieved [18].

There are many techniques to measure the similarity between the user query and the retrieved document [16].

I. *Cosine Similarity Measure (CSM):* is a popular method for calculating the similarity as a function of the angle made by the vectors. In CSM the sets of documents and queries are viewed as vectors, D and Q respectively. For each document, d element of D and query q element of Q, the cosine similarity is computed as follows:

$$CSM(q,d) = \frac{q \bullet d}{|q| * |d|}$$

II. *Jaccard Similarity Measure (JSM):* It measures similarity between the two documents. A commonly used measure of overlap of two sets of document. The JSM is computed as follows:

$JSM = 2\frac{|Q \cap D|}{|Q| + |D|}$, where Q = Queries and D= Documents.

III. *Dice Coefficient Measure (DCM):* It is used to compare the similarity between two samples of text DCM is computed as:

$DCM = \frac{|Q \cap D|}{|Q \cup D|}$, where Q = Queries and D= Documents.

2.3 Related works

The open source tools used to automate the acceptance testing in Java based code. Fixture code provides traceability between the requirements/tests and java code of the system under design. The connection between the specification (requirements/tests) and fixture code is achieved by using Concordion HTML tags. When programming tests and writing test fixture code we use Concordion as a library and simply follow the template. The requirements written as HTML files and acceptance tests written in Java using Concordion library are all part of the Java project set of files, which is in this case handled by Netbeans [27].

Autotestbot is one of the acceptance test tools proposed to automatically take as input user acceptance test cases written in natural language (English) and generate as output test code using a keyword driven test framework built on the top of Selenium Web driver framework. The written business user acceptance tests and the linguistic rules can be derived using Natural Language Processing (NLP) techniques. This approach consists of three phases. The first two phases consist of semi-automated steps that need to be done only once for a specific domain. These are used to setup the test framework and get it ready for use. The third phase consists of several automated steps that take in input acceptance tests written in natural language and generates test code that runs on Selenium Web driver. They build a tagged repository of user acceptance test cases in the first of the two setup phases. In the last phase, the Autotestbot Test system first takes as input the acceptance tests written by business users and uses the trained "test cases corpus" to parts of speech tag the input tests [7].

An empirically characterizing the contribution of acceptance tests to the clarification of the requirements coming from the customer. They focused on Fit (Framework for Integrated Test) which is an open source framework used to express acceptance test cases and a tool for improving the communication between analysts and developers. Fit table is a way to express acceptance tests, which can be automatically translated into executable test cases. Fit lets analysts write acceptance tests in the form of tables using simple HTML or even spreadsheets. The experimental results of

this Fit approach helps in the understanding of requirements without requiring a significant additional effort [29].

GuideGen tool is an approach that is used for requirement change and facilitates maintaining the requirements and acceptance tests alignment and communicating changes when requirements evolve. This tool-supported approach combines information retrieval and natural language processing concepts to provide suggestions about how to adapt the affected acceptance tests when their requirements change. This approach contributes the insights into requirements and acceptance tests management in industry. This tool automatically generates guidance in natural language on how to modify impacted acceptance tests when a requirement is changed [2].

The integrated feedback system is aims at improving feedback processes to enhance acceptance testing in distributed software project. To evaluate how a feedback system can affect acceptance testing in distributed projects, they chose a specific Feedback System(FS). The FS supports different platforms by providing clients for applications on the Web, and in Eclipse and Android frameworks. It allows the user to draw on a web page, take a screenshot of the result, add comments, and submit this feedback to a ticket system in the back-end of the feedback system [19].

Table 2.1 shows the summary of existing acceptance testing tools based on different measures.
Table 2.1 : Comparison of existing acceptance testing tools [7]

Tool	Approach/ Model used	Maintainability	Free form test cases	Reusability	Modularity easy to use	Consistency test case execution	Complexity for non programmers
Record playback	Selenium	___	_____	_____	_____	Low	Easy to use
Cucumber	RSL	✓	_____	✓	✓	High	complex
Robot Test Framework (RTF)	Model-based Testing (MBT)	✓	_____	✓	✓	High	Medium
Autotestbot	NLP	✓	✓	✓	✓	High	Easy to use

CHAPTER THREE: PROPOSED APPROACH

We proposed the semi-automated acceptance testing by using IR approach to help identify the features that need to be tested during acceptance testing. The existing related researches are not trying IR based to check the acceptance test. Requirements are expressions of user/customer needs while testing increases the likelihood that those needs are actually satisfied.

The proposed approach aims to support non-expert requirement analysts to accept software products that are ready for delivery. This approach aims to semi-automating the acceptance testing. In particular, the approach aims to save time and challenges of potential feature identification in complex software systems for acceptance testing. In this approach, we propose to use the two main documents available at the beginning and end of software development, i.e., software requirements specification and user manuals. The rationale for choosing these documents is that they usually need to be SRS prepared for the software system development to start and delivered for users and, hence, are almost always available. Figure 3.1 shows the overview of the proposed approach. Below we describe each step in.

Acceptance tests are developed from requirements artifacts. As a result of these, we select two artifacts (SRS and user's manual) that have similarities and are found at the beginning and last life cycle in software development. The process of determining what to say in a user's manual is the same as the process of eliciting requirements while writing use cases in SRS. Writing a good user's manual for a software product forces focusing on the user's view of the software.

In the proposed approach, the input data consists of the use case from SRS and the feature description from the user's manual document. To construct a corpus that suits for IR method, preprocessing of the input document is required. Both the SRS and user manual need to be broken up into proper granularity levels to define documents. These software artifacts are extracted to the defined granularity level, then they are pre-processed and represented as a set of documents in the resulting document and query.

Figure 3.1 shows overview of the proposed approach.

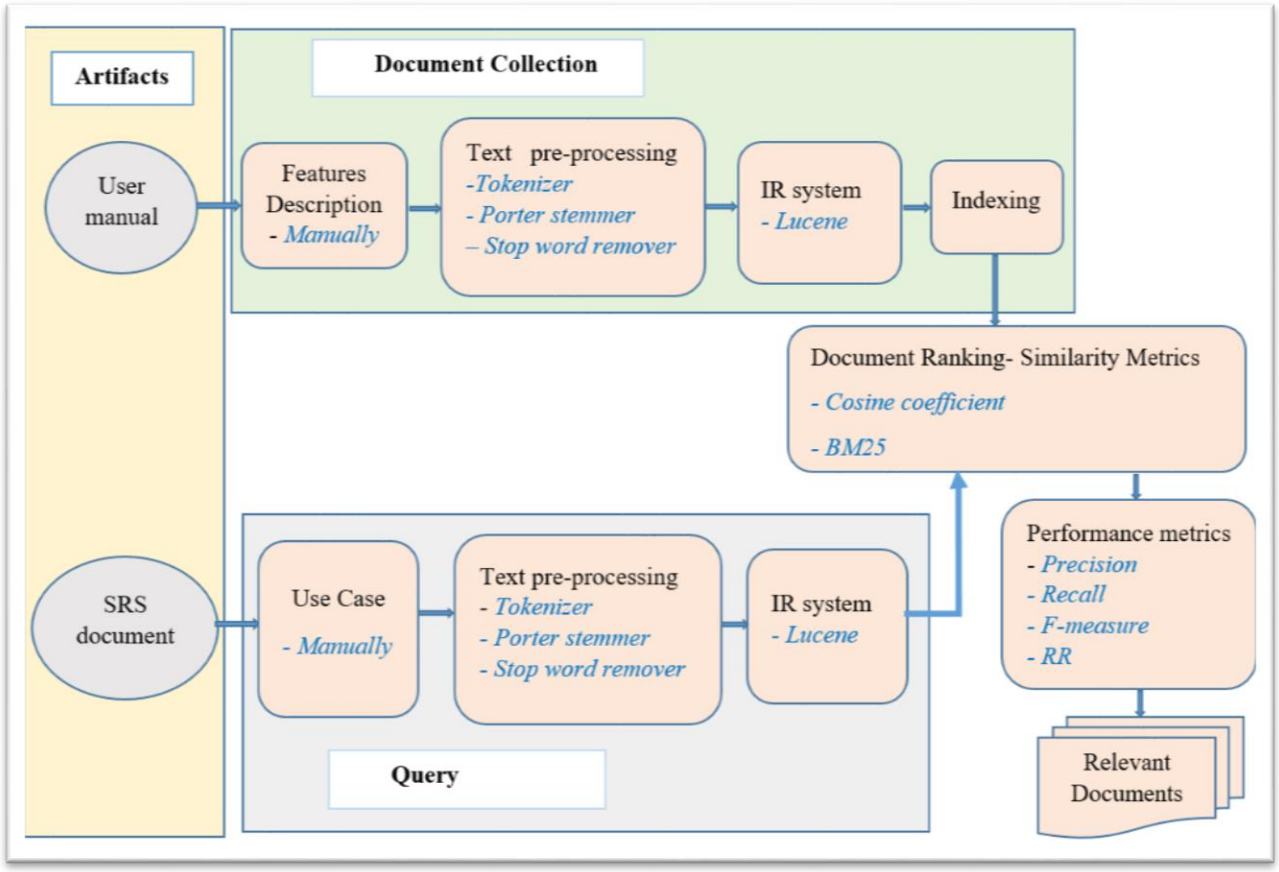


Figure 3.1 : Overview of proposed approach

The proposed approach aims to see if a use case defined in SRS is implemented in a software system or not. In our approach, we consider that each use-case corresponds to a feature in the developed software system. Requirement analyst develop a test description file in text format. For each use case the customer specifies acceptance tests as well.

A feature represents a characteristic or property of a system that is relevant to some user or stakeholder. Use case is related to a software artifact that defines a feature and behavior of that feature in software. Use cases are descriptions of external functions of a system from the viewpoint

of the users and external systems. Our approach provides a name and a description for each feature description based on a use-case name and description.

In this approach, each use-case name and description represents a query and each feature implementation represents a document. To identify a feature that corresponds to a use case, we use textual similarity between use-cases and feature descriptions. This similarity measure is calculated using cosine and BM25 in a lucene package and library. We utilize VSM to measure similarities and identify which use-cases best characterize the name and description of each feature implementation.

Since the information that is in a properly written user's manual for a software project is precisely what should be in an SRS of the Software.

Given the list of FD= (FD_1, FD_2, FD_3…,FDm) of lower-level user's manual elements and lists of use case UC=(UC_1, UC_2, UC_3,..) we map both UC and FD during acceptance testing in this approach.

We get, Acceptance testing= UC----> FD.

The user's manual allows the user to imagine interacting with the software, it serves to provide feedback on whether the software incorporates the required features of the software. The key advantage of user manuals over SRSs is the ease of validating the requirements document with the customer and users [28]. The SRS describes only what the system being specified does, while the user's manual describes conversations between the user and the system to achieve the user's goals.

3.1 Preparing the Corpus

In the document collection step in figure 3.1, the documents are indexed based on a vocabulary that is extracted from the documents themselves. The terms extracted from the documents are stored in an m × n matrix (term-by-document matrix).

Documents could be software artifacts such as source code, design documents, software requirements specifications (SRS), test cases, bug reports or user manuals. In the scenario we

considered, the requirements of the software system are prepared by the requirement engineers and given to a third party software developer as documents. The developer is expected to develop the software system and deliver it along with the user manual. To accept the developed software system, the engineers perform acceptance testing using the requirement document submitted at the very beginning of the development process. Hence, this approach proposes to use the SRS and user manual documents to facilitate the acceptance testing process by providing candidate features that needs to be tested to check a given requirement in the SRS.

One software system usually has only one SRS and manual documents. However, these documents have different sub-sections and could be re-organized at different granularity levels as documents. For our study, we considered a use case of the SRS to be one query document while the feature description in the user manuals are considered as one document in the corpus. The rationale for considering the use cases in the SRS and feature descriptions in the manual as document is that these granularity levels correspond to a specific feature in the software system that needs to be tested. The use cases are also the smallest units of SRS that represent a complete feature.

3.2 Text Preprocessing

Text preprocessing is a method to clean the text data and make it ready to feed data to the model. Once the documents are prepared at the granularity level of use cases of the SRS document and features of the user manuals, we conducted text normalization. The text normalization follows the following steps.

3.2.1 Tokenization

In this phase, we splitting the documents, phrases or paragraph into word tokens. We extracted terms/tokens from texts and convert capital letters into lower letters which is used to index a set of documents. This is important because the meaning of the text could easily be interpreted by analyzing the words present in text.

3.2.2 Stop-words Removal

We may omit common words. Stop-words are non-informative words such as conjunctions and articles that appear often in the documents and can be ignored. These include articles, punctuation, and numbers. A stop word function and/or a stop word list are applied to discard such words.

3.2.3 Stemming

It reduces each word to its common grammatical root by removing prefixes and suffixes. This allows the retrieval of artifacts that contain syntactic variations of the words in the requirements. In this thesis, these three steps will be done using Lucene libraries and NLTK tool kit which is written in python.

3.3 IR System

It helps to determine how relevant the document representation is to the query. To create the IR model, we use TF-IDF based VSM model in this study. Vector space model with TF-IDF will be used to compute the similarity between each pair of artifacts. In this method, the query and the document are represented in the form of vectors of keyword weights. The weights are computed using TF-IDF. VSM represents the textual content of requirements documents and queries extracted for the SRS as vectors in an n-dimensional space.

3.4 Document Indexing

The artifact indexing process aims to build a data structure that will allow quick searching of the text, making it easier to match a query with a document. Query and document should be represented using the same units/terms. We have Lucene support tool to index the documents.

3.5 Document Ranking

A document can be ranked based on its similarity. A list of matching documents for each query are ordered by decreasing similarity. In VSM source artifacts (SRS) are ranked against target artifacts (User manual) by computing a distance function (cosine similarity) between the corresponding vectors. On Lucene, we generate the candidate set and calculate the features against a query using BM25 textual similarity. BM25 improves upon TF*IDF. BM25 stands for “Best Match 25”. BM25 is a bag-of-words ranking function implementation. It is a ranking function that

ranks a set of documents based on the query terms appearing in each document. In this thesis, we used the top rank approach (also called cut point) types of threshold, without considering the actual values of similarity measures.

A similarity metrics is a function that computes the degree of similarity between a pair of vectors or documents. The similarity between each document in the document collection and the given query is computed using a cosine similarity in this thesis. The similarity between documents is typically measured by the cosine of the corresponding vectors, which increases as more terms are shared. In this thesis, the similarity of documents is compared and ranked by computing a distance function based on the cosine of the angle across the corresponding vectors which is calculated in lucene search engine libraries.

3.6 Evaluation Metrics in IR

An evaluation metric quantifies the performance of a specified IR model. There are different metrics in IR to evaluate the performance of retrieving documents. In this thesis, we identify four types of IR metrics to measure the performance of retrievals. These are Recall, Precision, F-measure and Reciprocal rank.

3.7 Formulating the Queries

To formulate the queries, we followed the same steps as those in corpus preparation. The queries in our approach are the use cases in the SRS. The use cases are selected as queries mainly because they are prepared first by the requirement engineer as the software requirements. Hence, they need to be checked if they are implemented accordingly.

These queries are compared with the ones associated with each document in the collection (user manual feature descriptions) to determine matches. Then, IR system generates the similarities between queries and documents. In our proposed approach, a query is created for each use case. This query contains the use-case name and its description.

CHAPTER FOUR: EXPERIMENT

4.1 Experimental setup

In this section, we explain four major steps that we followed to setup the experiment. The steps are:

Step 1: Preparing the corpus

Step 2: Text pre-processing.

Step 3: Document Ranking

Step 4: Evaluation Metrics

4.1.1 Dataset Description

For the case study, we selected two software systems. One of the software systems is proprietary while the other one is [open-source 23] [24]. The proprietary software system is taken from ethio-telecom. To select the software systems, we considered the availability of SRS and user manual documents as criteria. The SRS document should have a use case description of the features to be implemented in the software system. The user manual should also contain "how-to" descriptions of the features implemented in the software system.

The dataset from Ethiotelecom contains Fifteen use cases and the user manual contains Fifteen features. The dataset from open-source contains Nine use cases and the user manual contains Nine features. We use VSM with TF-IDF and cosine similarity to measure similarities between use cases and the features in the manuals. The use cases are used as queries while the features in the manuals are considered as documents to which we want to map the queries to.

Table 4.1 provides a concise overview of the main elements of the dataset that will be used in the experiment.

Table 4.1: A set of corpus formulated

Dataset One	Use Case(UC)	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15
	No. of words before preprocessing	41	29	40	29	51	23	23	26	20	22	41	32	21	12	36
	No. of token after preprocessing	30	27	33	23	47	20	17	21	20	16	34	29	20	11	31
	Feature Description(FD)	AC	RL	AM	AS	BS	MT	HR	EH	LG	AP	CO	NC	LS	RH	CH
	No. of words before preprocessing	38	28	37	29	52	21	23	22	15	38	33	22	13	39	20
	No. of token after preprocessing	33	26	32	22	44	15	20	21	14	34	28	19	11	31	18

Dataset Two	Use Case(UC)	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
	No. of words before preprocessing	75	79	37	40	73	69	20	55	40
	No. of token after preprocessing	62	65	22	29	54	53	15	44	27
	Feature Description(FD)	UM1	UM2	UM3	UM4	UM5	UM6	UM7	UM8	UM9
	No. of words before preprocessing	53	38	78	71	86	78	39	78	20
	No. of token after preprocessing	41	26	63	54	65	66	28	56	14

4.1.2 Pre-processing Phase

The SRS document and user manuals are splitted into smaller documents that contain use case and feature descriptions, respectively. For creating the queries, we used one use case description per query document while one feature description is used in one document to create the corpus. We applied the following pre-processing steps on both the use case and feature descriptions variants:

- Filtered numeric, punctuation and special characters.
- Filtered stop-words.
- Performed word stemming.

In this experiment, we used the tools implemented in the NLTK to perform pre-processing. To index the corpus and compute similarity, the Lucene search engine implemented for Eclipse SDK is used.

4.1.3 Document Ranking

We applied TF-IDF based VSM on the corpora, resulting in a list of ranked documents according to their similarity between query and document. The similarity between use cases and feature descriptions is measured as the cosine angle between the corresponding vectors, which increases as more terms are shared. This is done by using Lucene libraries and packages. The query and features are considered similar when the cosine similarity value is above a threshold. The threshold value used in this experiment is 0.70. We use this threshold after having tested many threshold values and other researchers also used this threshold value [3].

4.1.4 Evaluation Metrics

To evaluate the proposed approach, the quality and performance of IR systems for acceptance testing approach are to be measured. The standard evaluation metrics in information retrieval revolve around the idea of relevant and non-relevant documents. We have to qualify the similarity between the set of documents retrieved and the set of relevant documents provided by the analysts. The quality of IR methods is measured by how well the documents returned match the user's expectations. It gives an estimation of the goodness of the retrieved documents.

In this thesis, we used recall, precision, F-measure and reciprocal rank to evaluate the proposed IR-based acceptance testing method for searching performance measures. The evaluation of the retrieval process is the same for both datasets.

4.2 Tool

For the experiment, we applied a Lucene search engine library which runs on Eclipse IDE. NLTK is another tool that allows analysts to preprocess the use cases and feature descriptions used as query and corpus, respectively.

Lucene is a full-featured, high-performance, scalable text search engine library. It has a Java library that adds text indexing and searching capabilities to an application. It provides full-text indexing and searching. Eclipse Lucene application is the Eclipse plugin used to index and search documents. This tool is written in java to perform different processes for each document and the whole of the dataset.

The following diagram shows the processes between the artifacts and Lucene.

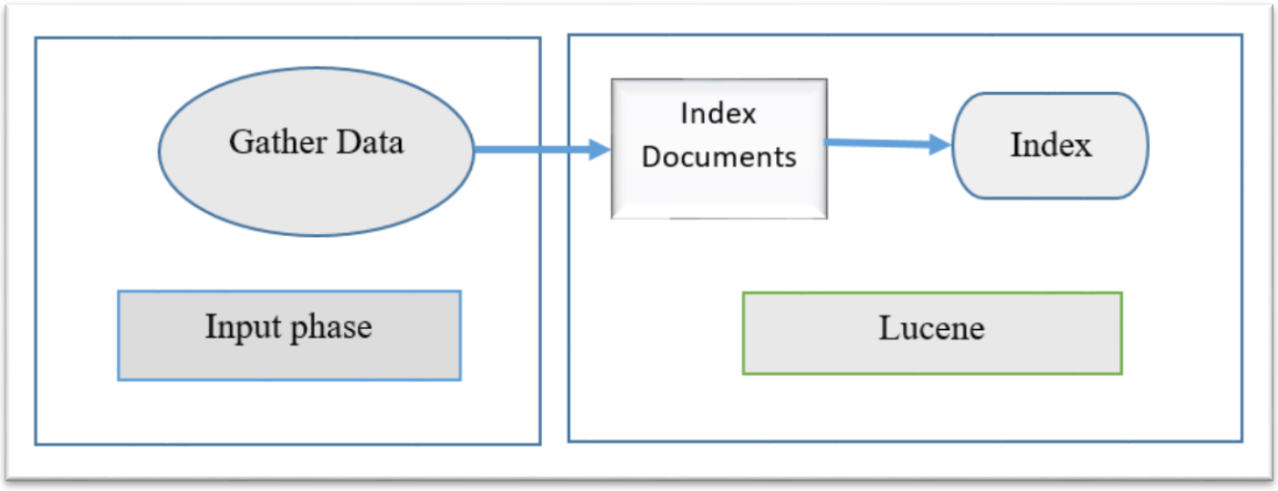


Figure 4.1 : Artifact integration with Lucene

The two main processes that are generated using the Lucene tool are indexing and searching. In the following subsection, we present the details of the two processes.

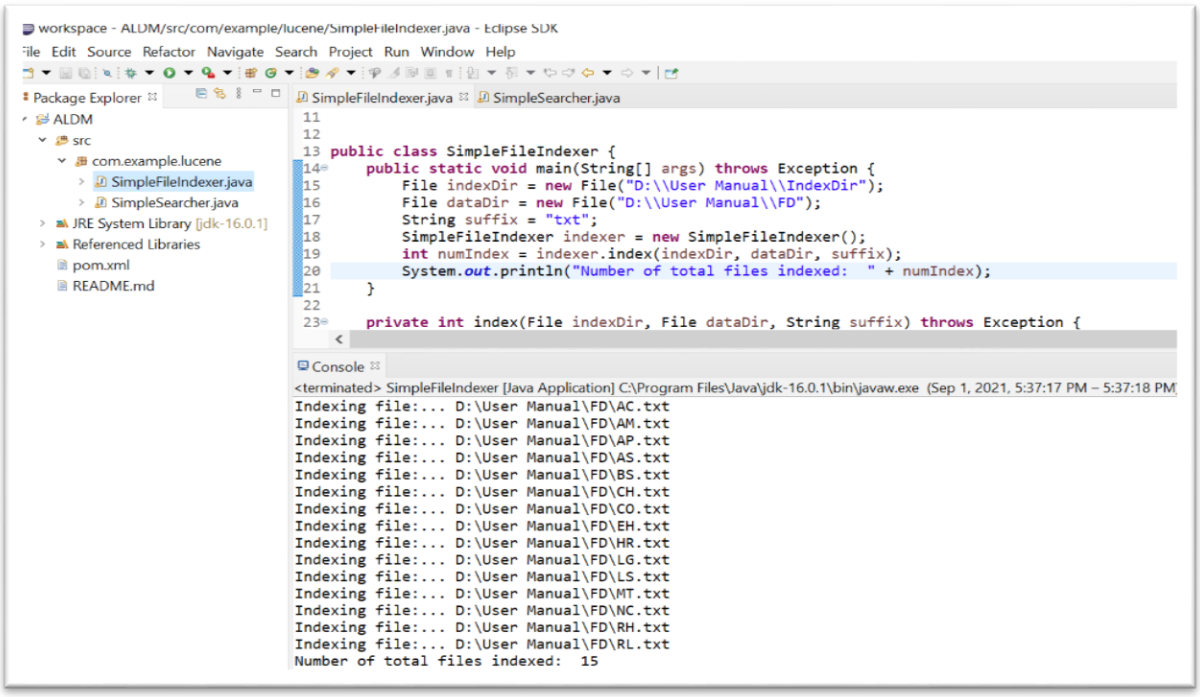
i. Indexing phase

Indexing is the central concept of any search engine. Indexing is the process of converting original data into an efficient lookup, which helps for rapid searching. The main purpose of indexing is to extract the meaningful keywords from the documents and represent them in a way that makes the process of finding relevant documents efficient [25]. As shown in figure 4.2, we indexed the extracted terms from SRS and the user's manual using the Lucene tool.

Indexer program needs two important command-line arguments.

- A path to a directory where Lucene index is to be stored--- "D:\\User manual \\IndexDir"
- A path to a directory which contains files to be indexed---- "D:\\User manual "

The below figure is the indexing of dataset One:



The below figure is the indexing of dataset two.

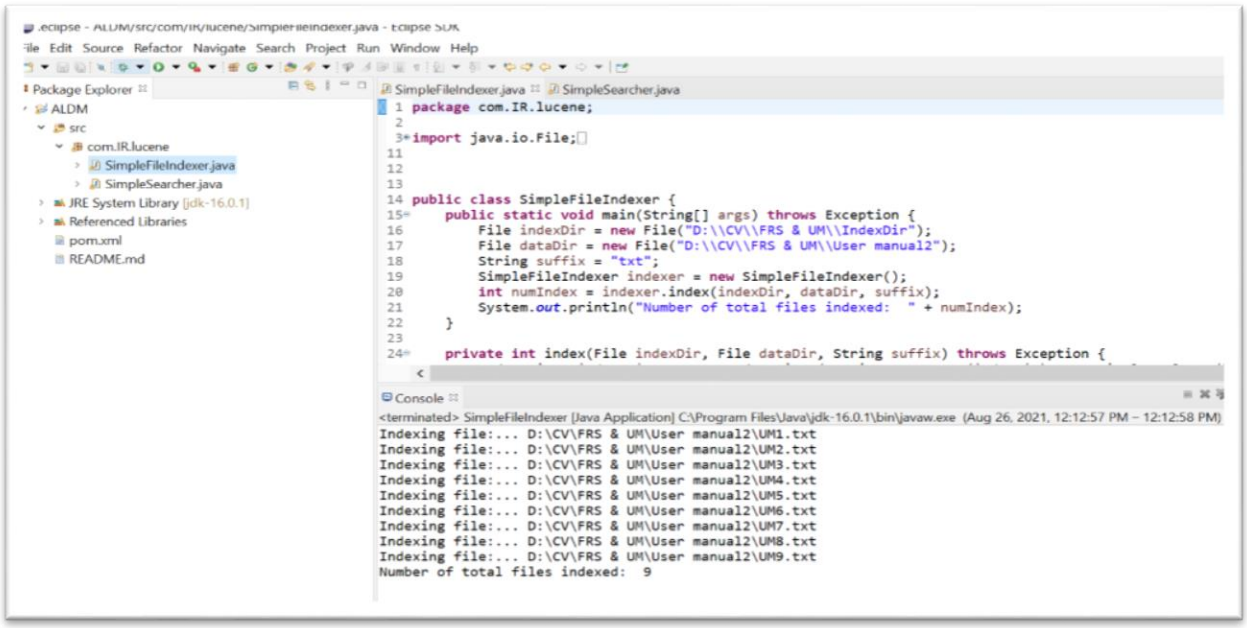


Figure 4.2 : Indexing documents

ii. Searching phase

Searching is a process of looking at words in an index to find out in which documents they appear in the file set. It does so by adding content to a full-text index. The content you add to Lucene can be from various sources, like a SQL/NoSQL database, a file system, or even from websites.

Searching requires an index to have already been built. It involves creating a Query (usually via a QueryParser) and handing this Query to an IndexSearcher, which returns a list of Hits. Lucene can achieve fast search responses because, instead of searching the text directly, it searches an indexed document. When searching, the cosine similarity of a query and each artifact is computed to determine the probability that an artifact is relevant to the query.

Figure 4.3 show that the searching processes in Lucene.

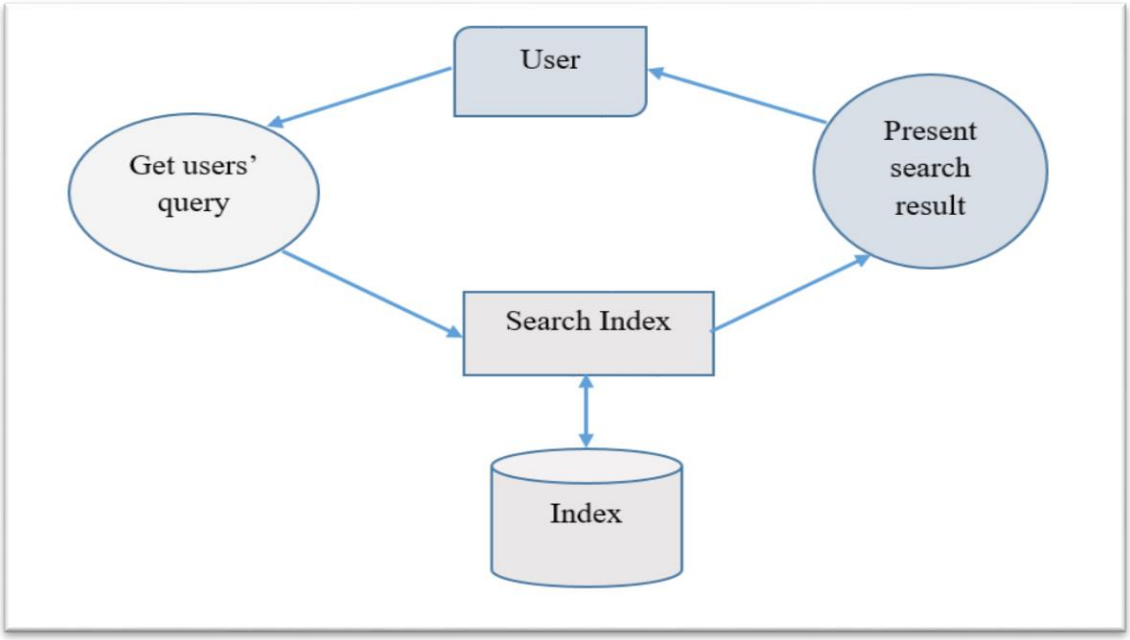


Figure 4.3 : Lucene Search Process Diagram [25]

The searcher program below needs two important command-line arguments.

- A path to the index created with Indexer--- "D:\\User manual \\IndexDir"
- A query to use to search the index---- "D:\\SRS Doc \\ Normalized\\ filename.txt"

The searcher program returns the documents that match the query in the form of Hits. It also prints a number of documents matched.

Figure 4.4 show that the sample of searching result for dataset two.

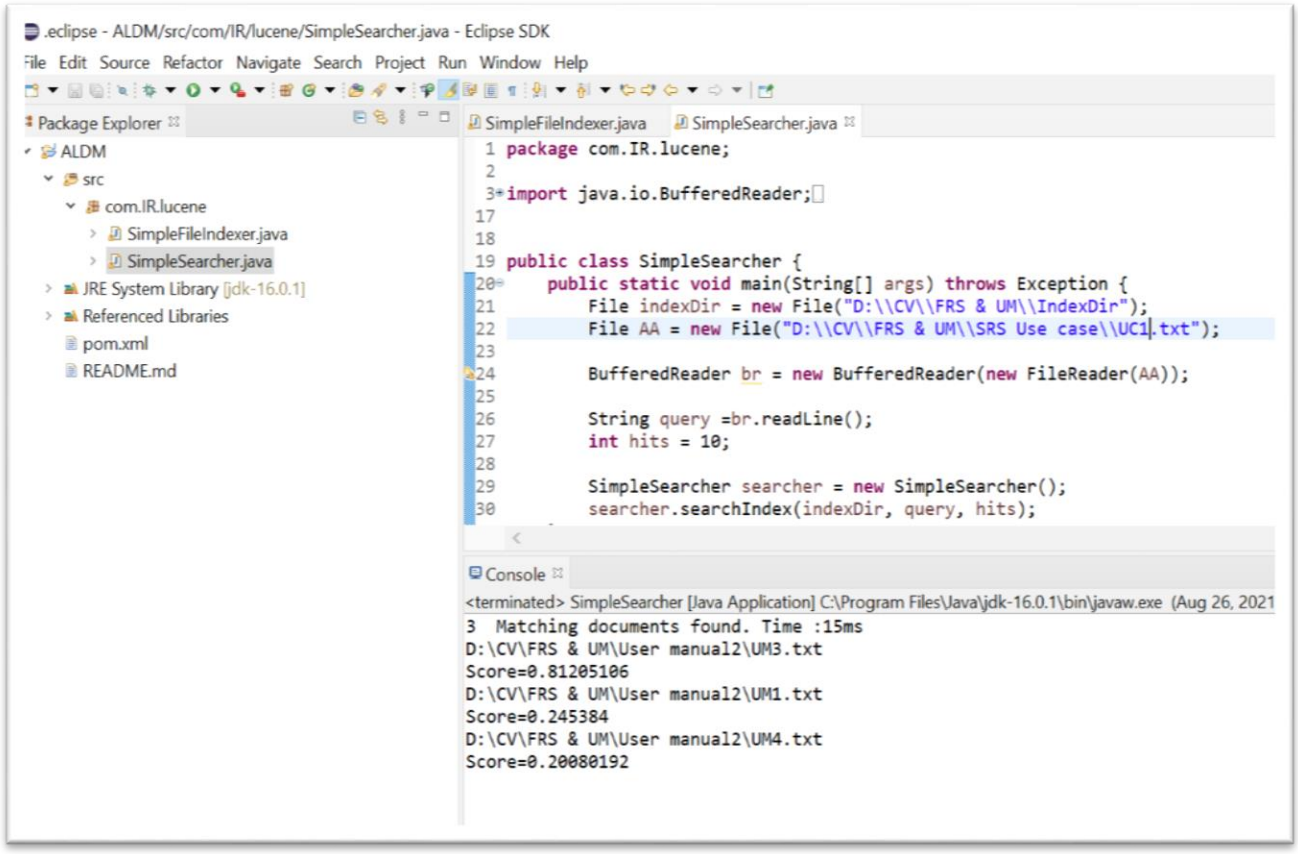


Figure 4.4: Sample of generated query result

iii. Generating the Results

The results returned by the search engine are a set of ranked documents that the tester inspects to decide their relevancy. The result is generated based on figure 4.3 processes. The documents are ranked based on their similarity to the query. The answer to the query is given in the form of a list of documents in descending order of their expected relevance to the query. Based on the figure 4.5 that generated the result of query (use case) in case of our approach, the following performances measures tables (4.4, 4.5 and 4.6) are generated.

4.3 Results and Discussion

This subsection discusses the results of our experiments by implementing the formulated corpus based on the set from Table 4.1, we computed the performance of the proposed approach based on the number of relevant documents found in this set and measured using recall, precision, F-measure and reciprocal rank metrics for each of the query in Tables 4.2 and 4.3.

Recall tells how many of the relevant documents retrieved from the total relevant documents, while precision tells how many of the retrieved documents are relevant. There is a tradeoff between recall and precision, and hence, as one increases the other decreases. F-measure is the harmonic mean of both precision and recall. Reciprocal rank is used to see where in the ranked list of retrieved documents is the relevant document ranked.

The results show that the proposed approach is able to identify all relevant documents and rank them first. The recall value of the result is 100%. This means that all relevant documents have been retrieved. If the value is 1 for precision, then all the retrieved documents are correct.

The precision and F-measure achieved are an average 30% and 44% respectively while RR is 1.00 for dataset one. In dataset two, the precision and F-measure achieved are an average 56% and 67% respectively while RR is 1.00. When the value of recall is 100%, UC and FD have one to one relationship. In these both case studies, the relation between UC and FD is one to one relation. For this reason, the value of recall is 100% for all queries. When the UC is implemented as two or more features in software product, the value of recall is different from 100%.

Table 4.2: IR Metrics of Dataset One

Dataset One		
Query Code	Precision	F-Measure
UC1	25%	40%
UC2	33%	50%
UC3	20%	33%
UC4	16%	28%
UC5	50%	67%
UC6	25%	40%
UC7	20%	33%
UC8	20%	33%
UC9	50%	67%
UC10	50%	67%
UC11	25%	40%
UC12	20%	33%
UC13	16%	28%
UC14	50%	67%
UC15	25%	40%
AV.	30%	44%

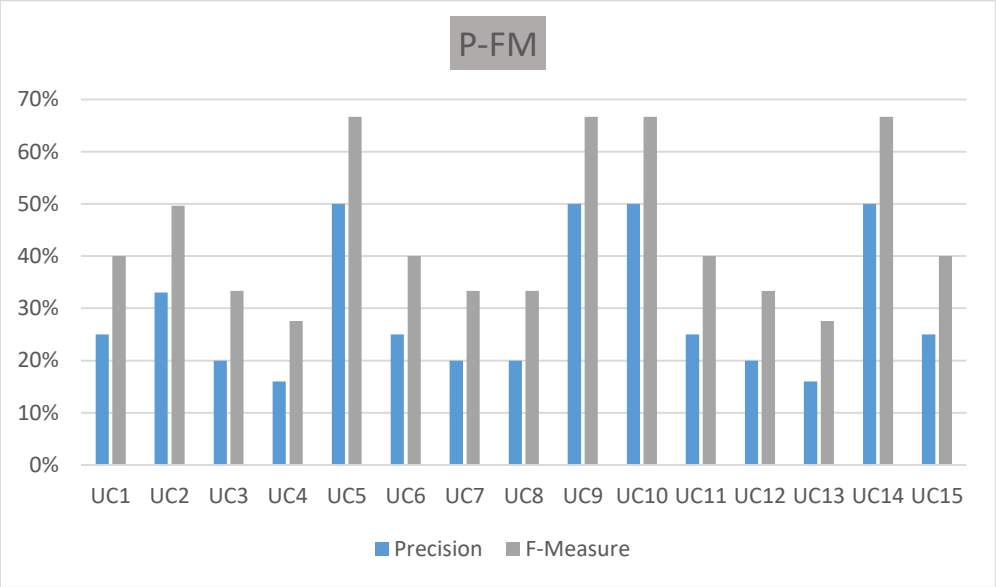


Figure 4.5: IR Performance Evaluation

Table 4.2 and Figure 4.5 show that IR performance measures such as; Precision, Recall, F-measure and Reciprocal rank for each query (use-case) of dataset one.

Table 4.3: IR Metrics of Dataset two

Dataset Two		
Query Code	Precision	F-Measure
UC1	33%	50%
UC2	100%	100%
UC3	100%	100%
UC4	25%	40%
UC5	50%	67%
UC6	25%	40%
UC7	25%	40%
UC8	50%	67%
UC9	100%	100%
AV.	56%	67%

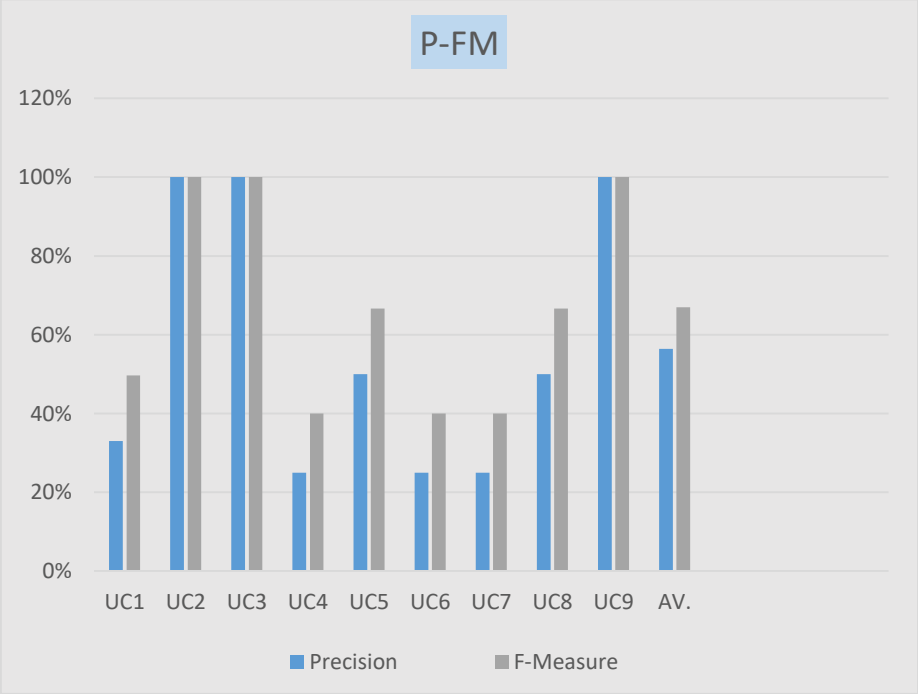


Figure 4.6: Performance Evaluation

Figure 4.6 and Table 4.3 show that the IR performance measures; Precision, Recall, F-measure, and Reciprocal rank for dataset two.

In both case studies, there is one-to-one relationships between use cases and feature descriptions. Hence, the recall result is 100%. In the general case, we may have one use case associated with several feature descriptions. The result of reciprocal rank is one for all queries. This is because the first relevant document was retrieved at first rank in both datasets. This shows that the IR system we used has a good performance of retrieving the relevant documents.

Discussion on Experimental Results

In our approach, we consider that each use-case corresponds to a feature description hence, the similarity between them helps us to use the two artifacts for acceptance testing.

Table 4.4: Similarity measures

Dataset One		
Query Code	Doc Code	Similarity
UC1	AC	0.85
UC2	RL	0.95
UC3	AM	0.82
UC4	AS	0.90
UC5	BS	0.71
UC6	CH	0.96
UC7	MT	0.86
UC8	HR	0.75
UC9	EH	0.90
UC10	LG	0.81
UC11	AP	1.00
UC12	CO	0.86
UC13	NC	0.97
UC14	LS	1.00
UC15	RH	1.00

Dataset Two		
Query Code	Doc Code	Similarity
UC1	UM3	0.81
UC2	UM5	0.96
UC3	UM6	0.98
UC4	UM7	0.97
UC5	UM8	0.96
UC6	UM4	0.97
UC7	UM9	0.99
UC8	UM1	0.71
UC9	UM2	0.78

As shown in Table 4.4 , those feature descriptions that have the highest similarity actually implement the UC described in the SRS are UC11 with AP, UC14 with LS and UC15 with RH which are found in dataset one. In our approach, the threshold value is below 0.70. This means that only pairs of use-cases and feature description having similarity greater than or equal to 0.70. This helps to identify those features that have strong similarity with use cases.

The results presented in Tables 4.2 and 4.3 show that all recall values are 100% indicating that all relevant documents are found by the proposed approach. This means that our approach efficiently associates use cases with their implementations of features in the software product. Precision values are [16% - 100%] in dataset one: similarity between a use-case and several features implementation is good. F-Measure values are consequently in [28% - 100%]. Precision values are in [25% - 100%] in dataset two: similarity between a use-case and several features implementation may be high. F-Measure values are consequently in [40% - 100%]. The Reciprocal rank values are 1 for all queries. This means that the most relevant feature was retrieved first. Hence, by using our approach, the requirement analysts can save their time and reduced the burden when compare with existing method of checking the location of requirement implementation in features. In addition to this, our approach is relatively easy to use compared to existing tools such as Robot Test Framework and Cucumber since we used natural language English for use cases and feature descriptions are allowed.

CHAPTER FIVE: CONCLUSION AND FUTURE WORK

In this section, we discussed it under two subsections. The first subsection is the conclusion of the study and the second subsection is discussed the future direction of the proposed tool.

5.1 Conclusion

Different commercial tools support software product acceptance testing. These tools, however, are expensive and require experts. To address these problems, in this thesis, we proposed IR based approach that automatically identifies candidate features to be tested for a given use case, and help non-expert software testers to easily conduct their work.

The proposed approach maps use cases and feature descriptions using IR methods. The IR methods used in the proposed approach are VSM and TF-IDF. We used cosine similarity to compute the similarity of two selected software artifacts (use cases and feature descriptions). We used the Lucene search engine and NLTK support software to perform different tasks in our experiment.

We have implemented and tested the proposed approach on two selected datasets. The results show that our approach has an average recall value of 100%, and average precision value of 30% and 56% for dataset one and two respectively. The average F-measure is 44% and 67% respectively. This shows that the approach is able to retrieved all relevant documents with the overhead of some false positives. The relevant documents found by the proposed approach are also ranked first. This indicates that testers would find the features that they need to test without spending much effort on searching. The similarity threshold used in the experiment is 0.7. This means that only pairs of use-cases and feature description having similarity greater than or equal to threshold are considered related.

5.2 Future Work

During the experiment and discussion phase, we had come up with the following ideas that can further enhance the capabilities of the approach.

- Compare the error-prone between manual and automate acceptance testing approach in software project development during delivery.
- We will improve the precision and relevance of our results by using adaptive similarity thresholds and semantic weighting schemes.

References

[1] *M. Unterkalmsteiner, T. Gorschek, R. Feldt, and E. Klotins, "Assessing requirements engineering and software test alignment - Five case studies," J. Syst. Softw., vol. 109, pp. 62–77, 2015.*

[2] *D. Der Wissenschaften, "Supporting Requirements and Acceptance Tests Alignment During Software Evolution," no. April, 2019.*

[3] K. Engineering and M. Huchard, "December 12, 2014 21:48," pp. 1–25, 2014.

[4] *S. Winkler and J. Von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," pp. 529–565, 2010.*

[5] *A. Espinoza and J. Garbajosa, "A study to support agile methods more effectively through traceability," pp. 53–69, 2011.*

[6] *S. K. Sundaram, J. H. Hayes, and A. Dekhtyar, "Baselines in requirements tracing," Proc. 2005 Work. Predict. Model. Softw. Eng. PROMISE 2005.*

[7] *A. Madhavan, "Semi Automated User Acceptance Testing using Natural Language Techniques," Iowa State Univ., 2014.*

[8] *Pinheiro, F.A.C,"Requirements traceability" In: Sampaio do Prado Leite, J.C., Doorn, J.H. (eds.) Perspectives on Software Requirements, pp. 93–113. Springer, Berlin (2003)*

[9] *J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Improving after-the-fact tracing and mapping: Supporting software quality predictions," IEEE Softw., vol. 22, no. 6, pp. 30–37, 2005.*

[10] *J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," Proc. IEEE Int. Conf. Requir. Eng., vol. 2003-January, pp. 138–147, 2003.*

[11] *Z. Li, M. Chen, L. G. Huang, and V. Ng, "Recovering traceability links in requirements documents," CoNLL 2015 - 19th Conf. Comput. Nat. Lang. Learn. Proc., pp. 237–246, 2015.*

[12] *M. Borg, "Advancing Trace Recovery Evaluation – Applied Information Retrieval in a Software Engineering Context," pp. 1–185, 2012.*

[13] *M. Shahid, "Test Coverage Measurement and Analysis on the Basis of Software Traceability Approaches," Int. J. Inf. Electron. Eng., no. January, 2011.*

[14] *A. D. E. Lucia, F. Fasano, and R. Oliveto, "Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods," vol. 16, no. 4, 2007.*

[15] *A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information Retrieval Methods for Automated Traceability Recovery," pp. 71–98, 2012.*

[16] *K. P. Reddy, T. R. Reddy, G. A. Naidu, and B. Vishnu, "Impact of Similarity Measures in Information Retrieval," pp. 54–59, 2018.*

[17] *A. Moffat and J. Zobel, "Rank-biased precision for measurement of retrieval effectiveness," ACM Trans. Inf. Syst., vol. 27, no. 1, 2008.*

[18] *R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to UML artifacts," Int. Work. Princ. Softw. Evol., pp. 49–54, 2004.*

[19] *O. Liskin, C. Herrmann, E. Knauss, T. Kurpick, B. Rumpe, and K. Schneider, "Supporting acceptance testing in distributed software projects with integrated feedback systems: Experiences and requirements," Proc. - 2012 IEEE 7th Int. Conf. Glob. Softw. Eng. ICGSE 2012, pp. 84–93, 2012.*

[20] *D. Maciel, A. C. R. Paiva, and A. R. Da Silva, "From requirements to automated acceptance tests of interactive apps: An integrated model-based testing approach," ENASE 2019 - Proc. 14th Int. Conf. Eval. Nov. Approaches to Softw. Eng., no. January, pp. 265–272, 2019.*

[21] *B. Haugset and G. K. Hanssen, "Automated acceptance testing: A literature review and an industrial case study," Proc. - Agil. 2008 Conf., no. September, pp. 27–38, 2008.*

[22] *J. Cleland-, B. Berenbach, and S. Clark, "Best Practices for Automated Traceability," no. June, pp. 27–35, 2007.*

[23] *A. Alhazmi et al., "Software Requirements Specification of the IUfA's UUIS -- a Team 4 COMP5541-W10 Project Approach," eprint arXiv:1005.0162, pp. 0–30, 2010.*

[24] *C. Science and S. Engineering, "User Manual of the UUIS A Team 4 COMP5541 - W10 Project Approach," 2010.*

[25] *S. Information, "Evaluation of Standard Information retrieval system related to specific queries," pp. 1–35.*

[26] *I. R. Introduction, "Algorithms for Information Retrieval – Introduction."*

[27] *T. Popović, "USING OPEN SOURCE TOOLS FOR AUTOMATED ACCEPTANCE TESTS AND REQUIREMENTS TRACEABILITY."*

[28] *D. M. Berry and T. Nelson, "User ' s Manual as a Requirements Specification : Case Studies," pp. 1–25, 2003.*

[29] *F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," Inf. Softw. Technol., vol. 51, no. 2, pp. 270–283, 2009..*

[30] *N. Nasir, "Acceptance Testing in Agile Software Development Perspectives from Research and Practice," 2021.*

[31] *M. Irshad, Assessing Reusability in Automated Acceptance Tests. 2018.*

Appendix

Semi Automated Acceptance Testing Using Information Retrieval

Debela Tadesse
Addis Ababa University,
Addis Ababa Institute of Technology,
Addis Ababa, Ethiopia
Debsse@gmail.com

Abstract — *Requirements Engineering(RE) is the discipline of eliciting, analyzing, specifying, validating and managing needs and constraints of a software product. Later, when the product is developed and delivered, stakeholders perform acceptance testing.*
Acceptance Testing is the verification conducted to use it that a software product provides expected behaviors, as expressed in requirements. One of the first tasks in acceptance testing is to know where a particular requirement is implemented in the software system. As the number of requirements increase, identifying where a particular requirement is implemented before the system becomes challenging and time-consuming, especially in environments where there is not so much expertise.
This thesis proposes an information retrieval (IR) based approach to address this problem. The approach uses the use cases in the SRS and the feature descriptions in the software manual, as a query and corpus, respectively, to identify where the use cases described in the SRS are implemented in the software system.
To evaluate the proposed approach, we conducted an experiment using two datasets collected from two software products. We observe that the recall values are 100% for all queries. The average value of F-measure is 44% and 67% in dataset one and two respectively. These indicate that, the proposed approach is able to associate use cases and features implementation. The Reciprocal rank values of all queries is 1, meaning that the most relevant features are found at first rank.
Hence, the software testers would easily identify the features to test a software product.
Keywords— *Acceptance Testing, Requirement Traceability, Information Retrieval, Software Requirement specification, software project, Query, Corpus, document, metric, Lucene.*

I. INTRODUCTION

Requirements Engineering(RE) is the discipline of eliciting, analyzing, specifying, validating and managing needs and constraints on a software product. It attempts to communicate the ideas and needs of the product's stakeholders to the engineers and developers who ultimately build the product. Successful RE requires a very good understanding of the business domain, the environment in which the system will be running, and the needs of the project's stakeholders [1].

Acceptance Testing is the verification that a software product provides expected behaviors, as expressed in requirements. It is a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it has met the required criteria for delivery to end-users. Acceptance testing is one of the final stages of testing, where the product is presented to the customer to validate if all the requirements are met, and the acceptance criteria for the product are satisfied [2].

Currently, many researchers proposed different approaches and tools to facilities acceptance testing. Concordion and FitNesse are open source tools used to automate acceptance testing for Java based software systems. The tools setup seems to be working well for small and medium size projects and the approach can be extrapolated to larger projects. The use of Concordion assumes that the requirements are written in native speaking language and kept in a set of HTML files [3].

Fit (Framework for Integrated Test) is an open source framework used to express acceptance test cases and a tool for improving the communication between analysts and developers. It lets analysts write acceptance tests in the form of tables (Fit tables) using simple HTML or even spreadsheets. A Fit table specifies the inputs and expected outputs for the test [4].

Cucumber or Robot Test Framework tools are constrained in that the user is forced to use the keywords pre-specified in their framework to create higher-level keywords. However, other research [14] uses natural language processing techniques to reduce

this overhead by not creating excessive higher-level keywords and instead allowing free form test cases to automated the acceptance testing. These tools require an extensive knowledge of the scripting language to create functional tests. Model-based Testing (MBT) technique is another approach that generates test cases from abstract representations of the system, named models, either graphical or textual. This approach uses RSL (Requirements Specification Language), where each Requirement is aligned with RSL Test Cases specifications and generates Test scripts that can be executed automatically by the Robot test automation tool over the System Under Test (SUT) [2].

The existing acceptance testing approaches usually target expert software testers. In different companies, the requirement analysis section is preparing the SRS to develop or adopt software projects. To accept the developed software product, they use a manual method of acceptance testing. This acceptance testing is done by using SRS documents and manually cross-checking to verify and validate the implementation of software requirements. As the complexity of software increases, using manual methods for acceptance testing becomes too challenging and time-consuming. Hence, the acceptance testing approach that supports non-expert requirement analyst to test requirement implementation is needed.

II. LITERATURE REVIEW

Nowadays, requirements engineering attempts to communicate the ideas and needs of the software product's stakeholders to the engineers and developers who ultimately build the software product. While stakeholders generally know the functions and features of the software should include, they have difficulty quantifying the fine-grain details and behavior of a system [1].

A. Acceptance Testing

Acceptance tests are defined as "customer tests,", "customer-inspired tests," and "conditions of satisfaction". In systems and software engineering it is defined as "formal testing conducted to enable a user, customer, or other authorised entity to determine whether to accept a system or component." It is also used to evaluate whether the customer requirements are met. An acceptance test is carried out by users who take the initiative in confirming whether the software under development implements the requirements of the users [5].

Acceptance Testing is an indication both to developers and to product owners that the software product satisfies the requirements specification. Requirements continue to be traced through user acceptance testing, ultimately demonstrating that the delivered solution conforms to the defined requirements and meets the needs of the client. The success of any software product lies with the fact that the customers/users accept it. It provides confidence that the delivered system meets the business requirements of the customer (and users) [3].

Acceptance tests are used to determine if a system performs according to contractual requirements. It is feature level tests, which ensure that the product functions according to defined criteria. It is extracted from the requirements and can be used inside organizations to communicate the implementation between different developers [8].

Acceptance testing can be used to track and evaluate the software development process by measuring the number of tests that pass or fail. It is used for comparing a system to its initial requirements and the current needs of its end-users and for verifying whether the system implements the requirements as intended. One of the main reasons for the failure of many software projects is the late discovery of a mismatch between the customers' expectations and the pieces of functionality implemented in the delivered system [15].

Automated acceptance relies on various information retrieval algorithms that use techniques such as the vector-space model or the probabilistic network model to compute the likelihood of a link based on the occurrence and distribution of terms. It also uses scripts and tools that prepare data and a state, then executes the steps required to verify the scenario in an automated way [16].

B. Information Retrieval

Information Retrieval (IR) is the process of discovering documents relevant to an information request. The main aim of IR is to identify relevant documents in document collections given user-specified information needs. In practice, the query and the searchable documents could be any type of software artifact in the system. Information Retrieval methods have been largely adopted to identify traceability links based on the textual similarity of software artifacts [11].

The IR approaches are generally based on chunks of natural language text. It returns a set of matching documents from a set of documents (corpus) for a given query. First, extracting key terms from each document in the corpus and then computing the similarity between the terms of the query and the key terms of each document. A list of candidates is then presented to the user who has to decide which of the candidates are relevant to his query [12].

The IR techniques mostly used to identify links in documents are Latent Semantic Indexing (LSI), Probabilistic Modeling (PM), and Vector Space Model (VSM). Below we briefly describe each technique [13].

IR systems compute for each document in a collection, a score that estimates the similarity between that document and a query. In typical systems, each score represents an estimated probability that the document is relevant to the information needed expressed by the query. The three main elements of IR system are; a document collection (corpus), a set of information needs (queries), and relevance judgments telling what documents are relevant to these information needs [10].

IR system retrieval performance can be evaluated by calculating the metrics are; Recall, Precision, F-measure and Reciprocal rank [9].

C. Related works

The open source tools used to automate the acceptance testing in Java based code. When programming tests and writing test fixture code, we use Concordion as a library and simply follow the template. The requirements written as HTML files and acceptance tests written in Java using Concordion library are all part of the Java project set of files, which is in this case handled by Netbeans [3].

Autotestbot is one of the acceptance test tools proposed to automatically take as input user acceptance test cases written in natural language (English) and generate as output test code using a keyword driven test framework built on the top of Selenium Web driver framework. The written business user acceptance tests and the linguistic rules can be derived using NLP techniques. The Autotestbot Test system first takes as input the acceptance tests written by business users and uses the trained "test cases corpus" to parts of speech tag the input tests [14].

Fit (Framework for Integrated Test) is an open source framework used to express acceptance test cases and a tool for improving the communication between analysts and developers. Fit table is a way to express acceptance tests, which can be automatically translated into executable test cases. Fit lets analysts write acceptance tests in the form of tables using simple HTML or even spreadsheets [4].

GuideGen tool is an approach that is used for requirement change and facilitates maintaining the requirements and acceptance tests alignment and communicating changes when requirements evolve. This tool-supported approach combines information retrieval and natural language processing (NLP) concepts to provide suggestions about how to adapt the affected acceptance tests when their requirements change [7].

Tool	Approach/ Model used	Maintainability	Free form test cases	Reusability	Modularity easy to use	Consistency test case execution	Complexity for non programmers
Record playback	Selenium	___	___	___	____	Low	Easy to use
Cucumber	RSL	✓	___	✓	✓	High	complex
Robot Test Framework (RTF)	Model-based Testing (MBT)	✓	___	✓	✓	High	Medium
Autotestbot	NLP	✓	✓	✓	✓	High	Easy to use

Table 2.1: Comparison of existing acceptance testing tools

III. PROPOSED APPROACH

We proposed acceptance testing that semi-automated by using IR method to help identify the features that need to be tested during acceptance testing. Requirements are expressions of user/customer needs while testing increases the likelihood that those needs are actually satisfied.

The proposed approach aims to support non-expert requirement analysts to accept software product that are ready for delivery. This approach aims to semi-automated the acceptance testing. In particular, the approach aims to save time and challenges of potential feature identification in complex software systems for acceptance testing. In this approach, we propose to use the two main documents available at the beginning and end of software development, i.e., software requirements specification and user manuals. The rationale for choosing these documents is that they usually need to be SRS prepared for the software system development to start and delivered for users and, hence, are almost always available. Figure 3.1 shows the overview of the proposed approach. Below we describe each step in.

In the proposed approach, the input data consists of the use case from SRS and the feature description from the user's manual document. To construct a corpus that suits for IR method, preprocessing of the input document is required. Both the SRS and user manual need to be broken up into proper granularity levels to define documents. These software artifacts are extracted to the defined granularity level, then they are pre-processed and represented as a set of documents in the resulting document and query.

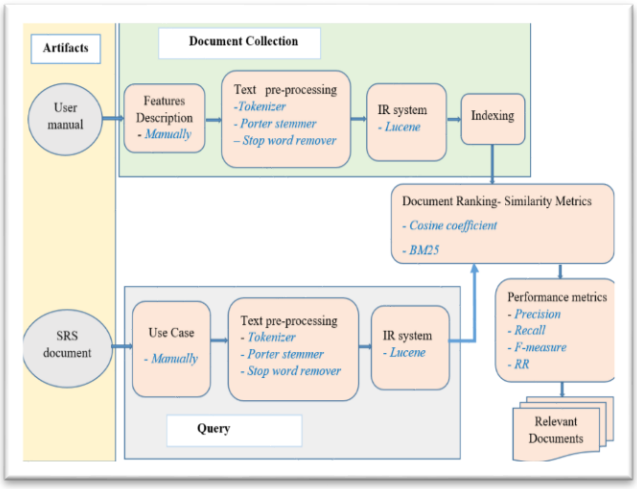


Fig. 3.1: Experimental Design

The proposed approach aims to see if a use case defined in SRS is implemented in a software system or not. In our approach, we consider that each use-case corresponds to a feature in the developed software system. Requirement analysts develop a test description file in text format. For each use case the customer specifies acceptance tests as well.

A feature represents a characteristic or property of a system that is relevant to some user or stakeholder. Use case is related to a software artifact that defines a feature and behavior of that feature in software. Use cases are descriptions of external functions of a system from the viewpoint of the users and external systems. Our approach provides a name and a description for each feature description based on a use-case name and description.

In this approach, each use-case name and description represents a query and each feature implementation represents a document. To identify a feature that corresponds to a use case, we use textual similarity between use-cases and feature descriptions. This similarity measure is calculated using cosine and BM25 in a lucene package and library. We utilize VSM to measure similarities and identify which use-cases best characterize the name and description of each feature implementation.

Since the information that is in a properly written user's manual for a software project is precisely what should be in an SRS of the Software.

Given the list of FD= (FD1, FD2, FD3…,FDm) of lower-level user's manual elements and lists of use

case UC=(UC1, UC2, UC3,..) we map both UC and FD during acceptance testing in this approach.

We get, Acceptance testing= UC----> FD.

IV. EXPERIMENT

A. Experimental setup

In this section, we explain four major steps that we followed to setup the experiment. The steps are:

Step 1: Preparing the corpus

Step 2: Text pre-processing.

Step 3: Document Ranking

Step 4: Evaluation Metrics

Dataset Description: For the case study, we selected two software systems. To select the software systems, we considered the availability of SRS and user manual documents as criteria. The SRS document should have a use case description of the features to be implemented in the software system. The user manual should also contain "how-to" descriptions of the features implemented in the software system.

The first dataset contains Fifteen use cases and the user manual contains Fifteen features, while the second dataset contains Nine use cases and the user manual contains Nine features. We use TF-IDF based VSM and cosine similarity to measure similarities between use cases and the features in the manuals. The use cases are used as queries while the features in the manuals are considered as documents to which we want to map the queries to.

Pre-processing Phase: The SRS document and user manuals are splitted into smaller documents that contain use case and feature descriptions, respectively. For creating the queries, we used one use case description per query document while one feature description is used in one document to create the corpus. We applied the following pre-processing steps on both the use case and feature descriptions variants:

- Filtered numeric, punctuation and special characters.
- Filtered stop-words.
- Performed word stemming.

In this experiment, we used the tools implemented in the NLTK tool kit to perform pre-processing. To index the corpus and compute similarity, the Lucene search engine implemented for Eclipse SDK is used. Descriptions corpus, then we used it as input to Lucene search engine on Eclipse SDK.

Document Ranking: We applied TF-IDF based VSM on the corpora, resulting in a list of ranked documents according to their similarity between query and document. The similarity between use cases and feature descriptions is measured as the cosine angle between the corresponding vectors, which increases as more terms are shared. This is done by using Lucene libraries and packages. The query and features are considered similar when the cosine similarity value is above a threshold. The threshold value used in this experiment is 0.70. We use this threshold after having tested many threshold values and other researchers also used this threshold value [6].

Evaluation Metrics: To evaluate the proposed approach, the quality and performance of IR systems for acceptance testing approach are to be measured. The standard evaluation metrics in information retrieval revolve around the idea of relevant and non-relevant documents. We have to qualify the similarity between the set of documents retrieved and the set of relevant documents provided by the analysts. The quality of IR methods is measured by how well the documents returned match the user's expectations. It gives an estimation of the goodness of the retrieved documents.

In this thesis, we used recall, precision, F-measure and reciprocal rank to evaluate the proposed IR-based acceptance testing method for searching performance measures. The evaluation of the retrieval process is the same for both datasets.

B. Tool

For the experiment, we applied a Lucene search engine library which runs on Eclipse IDE Eclipse. NLTK is another tool that allows analysts to preprocess the use cases and feature descriptions used as query and corpus, respectively.

Lucene is a full-featured, high-performance, scalable text search engine library. It has a Java library that adds text indexing and searching capabilities to an application. It provides full-text indexing and searching. Eclipse Lucene application is the Eclipse plugin used to index and search documents. This tool is written in java to perform different processes for each document and the whole of the dataset.

The two main processes that are generated using the Lucene tool are indexing and searching. In the following subsection, we present the details of the two processes.

Indexing phase: is the central concept of any search engine. Indexing is the process of converting original data into an efficient lookup, which helps for rapid searching. The main purpose of indexing is to extract the meaningful keywords from the documents and represent them in a way that makes the process of finding relevant documents efficient. As shown in figure 4.2, we indexed the extracted terms from SRS and the user's manual using the Lucene tool.

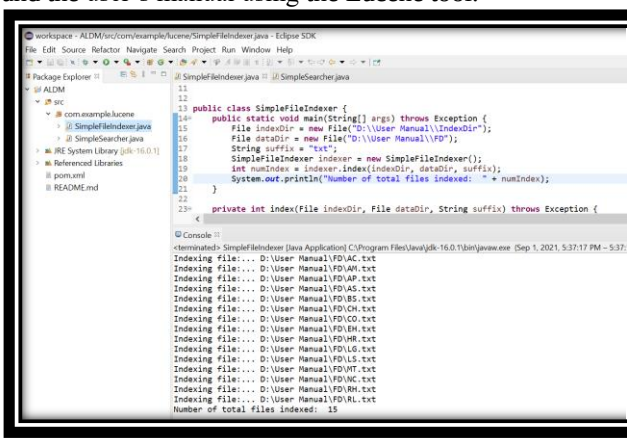


Fig. 4.1: Indexing Result

Searching phase: is a process of looking at words in an index to find out in which documents they appear in the file set. It does so by adding content to a full-text index. The content you add to Lucene is from a file system. Searching requires an index to have already been built. When searching, the cosine similarity of a query and each artifact is computed to determine the probability that an artifact is relevant to the query.

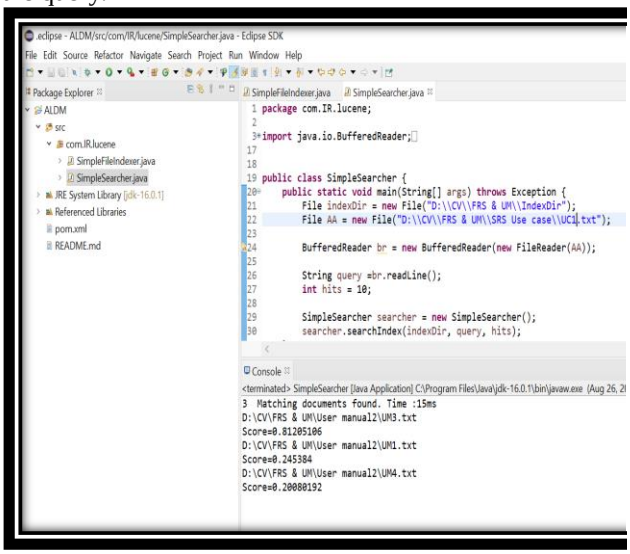


Fig. 4.2: Searching Result

Generating the Results: The results returned by the search engine are a set of ranked documents that the tester inspects to decide their relevancy. The documents are ranked based on their similarity to the query. The answer to the query is given in the form of a list of documents in descending order of their expected relevance to the query.

C. Experimental Results

We report the results of experimentation based on two datasets. We computed the performance of the proposed approach based on the number of relevant documents found in this set and measured using recall, precision, F-measure and reciprocal rank metrics for each of the query in Tables 4.1.

Recall tells how many of the relevant documents retrieved from the total relevant documents, while precision tells how many of the retrieved documents are relevant. There is a tradeoff between recall and precision, and hence, as one increases the other decreases. F-measure is the harmonic mean of both precision and recall. Reciprocal rank is used to see where in the ranked list of retrieved documents is the relevant document ranked.

The results show that the proposed approach is able to identify all relevant documents and rank them first. The recall value of the result is 100%. This means that all relevant documents have been retrieved. If the value is 1 for precision, then all the retrieved documents are correct.

The precision and F-measure achieved are an average 30% and 44% respectively while RR is 1.00 for dataset one. In dataset two, the precision and F-measure achieved are an average 56% and 67% respectively while RR is 1.00. This means that when the value of recall is 100%, UC and FD have one to one relationship.

Dataset One		
Query Code	Precision	F-Measure
UC1	25%	40%
UC2	33%	50%
UC3	20%	33%
UC4	16%	28%
UC5	50%	67%
UC6	25%	40%
UC7	20%	33%
UC8	20%	33%
UC9	50%	67%
UC10	50%	67%
UC11	25%	40%
UC12	20%	33%
UC13	16%	28%
UC14	50%	67%
UC15	25%	40%
AV.	30%	44%

Dataset One		
Query Code	Precision	F-Measure
UC1	25%	40%
UC2	33%	50%
UC3	20%	33%
UC4	16%	28%
UC5	50%	67%
UC6	25%	40%
UC7	20%	33%
UC8	20%	33%
UC9	50%	67%
UC10	50%	67%
UC11	25%	40%
UC12	20%	33%
UC13	16%	28%
UC14	50%	67%
UC15	25%	40%
AV.	30%	44%

Table 4.1: IR performance metrics

In both case studies, there is one-to-one relationships between use cases and feature descriptions. Hence, the recall result is 100%. In the general case, we may have one use case associated with several feature descriptions. The result of reciprocal rank is one for all queries. This is because the first relevant document was retrieved at first rank in both datasets. This shows that the IR system we used has a good performance of retrieving the relevant documents.

Discussion on Experimental Results: In our approach, we consider that each use-case corresponds to a feature description hence, the similarity between them helps us to use the two artifacts for acceptance testing.

Dataset One		
Query Code	Doc Code	Similarity
UC1	AC	0.85
UC2	RL	0.95
UC3	AM	0.82
UC4	AS	0.90
UC5	BS	0.71
UC6	CH	0.96
UC7	MT	0.86
UC8	HR	0.75
UC9	EH	0.90
UC10	LG	0.81
UC11	AP	1.00
UC12	CO	0.86
UC13	NC	0.97
UC14	LS	1.00
UC15	RH	1.00

Dataset Two		
Query Code	Doc Code	Similarity
UC1	UM3	0.81
UC2	UM5	0.96
UC3	UM6	0.98
UC4	UM7	0.97
UC5	UM8	0.96
UC6	UM4	0.97
UC7	UM9	0.99
UC8	UM1	0.71
UC9	UM2	0.78

Table 4.2: Similarity measures

As shown in 4.2 Table, those feature descriptions that have the highest similarity actually implement the UC described in the SRS are UC11 with AP, UC14 with LS and UC15 with RH which are found in dataset one. In our approach, the threshold value is below 0.70. This means that only pairs of use-cases and feature description have similarity greater than or equal to 0.70. This helps to identify those features that have strong similarity with use cases.

The results presented in Tables 4.2 and 4.3 show that all recall values are 100% indicating that all relevant documents are found by the proposed approach. This means that our approach efficiently associates use cases with their implementations of features in the software product. Precision values are [16% - 100%] in dataset one: similarity between a use-case and several features implementation is good. F-Measure values are consequently in [28% - 100%]. Precision values are in [25% - 100%] in dataset two: similarity between a use-case and several features implementation may be high. F-Measure values are consequently in [40% - 100%]. The Reciprocal rank values are 1 for all queries. This means that the most relevant feature was retrieved first. Hence, by using our approach, the requirement analysts can save their time and reduce the burden.

V. CONCLUSION AND FUTURE WORK

A. Conclusion

Different commercial tools support software product acceptance testing. These tools, however, are expensive and require experts. To address these problems, in this thesis, we proposed an IR based approach that automatically identifies candidate features to be tested for a given use case, and help non-expert software testers to easily conduct their work.
The proposed approach maps use cases and feature descriptions using IR methods. The IR methods used in the proposed approach are VSM and TF-IDF. We used cosine similarity to compute the similarity of two selected software artifacts (use cases and feature descriptions). We used the Lucene search engine and NLTK support software to perform different tasks in our experiment.

We have implemented and tested the proposed approach on two selected datasets. The results show that our approach has an average recall value of 100%, and average precision value of 30% and 56% for dataset one and two respectively. The average F-measure is 44% and 67% respectively. This shows that the approach is able to retrieve all relevant documents with the overhead of some false positives. The relevant documents found by the proposed approach are also ranked first. This indicates that testers would find the features that they need to test without spending much effort on searching. The similarity threshold used in the experiment is 0.7. This means that only pairs of use-cases and feature description having similarity greater than or equal to threshold are considered related.

B. Future Work

During the experiment and discussion phase, we came up with the following ideas that can further enhance the capabilities of the approach.

- Compare the error-prone between manual and automate acceptance testing approach in software project development during delivery.
- We will improve the precision and relevance of our results by using adaptive similarity thresholds and semantic weighting schemes.

REFERENCES

[1] *H. Elshandidy and S. Mazen, "Agile and Traditional Requirements Engineering: A Survey," vol. 4, no. 9, pp. 473–482, 2013.*

[2] *D. Maciel, A. C. R. Paiva, and A. R. Da Silva, "From requirements to automated acceptance tests of interactive apps: An integrated model-based testing approach," ENASE 2019 - Proc. 14th Int. Conf. Eval. Nov. Approaches to Softw. Eng., no. January, pp. 265–272, 2019.*

[3] *T. Popović, "USING OPEN SOURCE TOOLS FOR AUTOMATED ACCEPTANCE TESTS AND REQUIREMENTS TRACEABILITY."*

[4] *F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," Inf. Softw. Technol., vol. 51, no. 2, pp. 270–283, 2009.*

[5] *N. Nasir, "Acceptance Testing in Agile Software Development Perspectives from Research and Practice," 2021.*

[6] K. Engineering and M. Huchard, "December 12, 2014 21:48," pp. 1–25, 2014.

[7] *D. Der Wissenschaften, "Supporting Requirements and Acceptance Tests Alignment During Software Evolution," no. April, 2019.*

[8] *M. Irshad, Assessing Reusability in Automated Acceptance Tests. 2018.*

[9] *J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," Proc. IEEE Int. Conf. Requir. Eng., vol. 2003-January, pp. 138–147, 2003.*

[10] *A. Moffat and J. Zobel, "Rank-biased precision for measurement of retrieval effectiveness," ACM Trans. Inf. Syst., vol. 27, no. 1, 2008.*

[11] *A. D. E. Lucia, F. Fasano, and R. Oliveto,*

"Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods," vol. 16, no. 4, 2007.

[12] *S. Winkler and J. Von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," pp. 529–565, 2010.*

[13] *Pinheiro, F.A.C,"Requirements traceability" In: Sampaio do Prado Leite, J.C., Doorn, J.H. (eds.) Perspectives on Software Requirements, pp. 93–113. Springer, Berlin (2003).*

[14] *A. Madhavan, "Semi Automated User Acceptance Testing using Natural Language Techniques," Iowa State Univ., 2014.*

[15] *B. Haugset and G. K. Hanssen, "Automated acceptance testing: A literature review and an industrial case study," Proc. - Agil. 2008 Conf., no. September, pp. 27–38, 2008.*

[16] *J. Cleland-, B. Berenbach, and S. Clark, "Best Practices for Automated Traceability," no. June, pp. 27–35, 2007.*