A PROCESS FRAMEWORK FOR MANAGING IMPLICIT REQUIREMENTS USING ANALOGY-BASED REASONING

EMEBO, ONYEKA CHIBUEZE

CU023020043

JULY, 2017

A PROCESS FRAMEWORK FOR MANAGING IMPLICIT REQUIREMENTS USING ANALOGY-BASED REASONING

BY

EMEBO, ONYEKA CHIBUEZE
B.Sc. and M.Sc. Computer Science
(Covenant University)
Matriculation Number: **CU023020043**

A Thesis Submitted to the Department of Computer and Information Sciences, College of Science and Technology, Covenant University, Ota, Nigeria. In Partial Fulfilment of the Requirements for the Award of Doctor of Philosophy (Ph.D) Degree in Computer Science.

JULY, 2017

ACCEPTANCE

This is to attest that this thesis is accepted in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy in Computer Science** in the Department of Computer and Information Sciences, College of Science and Technology, Covenant University, Ota.

PHILIP JOHN AINWOKHAI _____

(Secretary, School of Postgraduate Studies) **Signature and Date**

PROFESSOR SAMUEL WARA _____

(Dean, School of Postgraduate Studies) **Signature and Date**

DECLARATION

I, **EMEBO, ONYEKA CHIBUEZE** declare that this research was carried out by me under the supervision of Professor Charles Korede Ayo of the Department of Computer and Information Sciences, Covenant University, Ota, Ogun State, Nigeria and Dr. Olawande Daramola of the Department of Computer and Information Sciences, Covenant University, Ota, Ogun State, Nigeria. I attest that this thesis has not been presented either wholly or partly for the award of any degree elsewhere. All the sources of materials consulted in the course of this academic research are duly acknowledged.

EMEBO, ONYEKA CHIBUEZE _____

Signature and Date

CERTIFICATION

We certify that the thesis titled "**A Process Framework for Managing Implicit Requirements using Analogy-Based Reasoning**" is an original research work carried out by **EMEBO, ONYEKA CHIBUEZE (CU023020043)** in the Department of Computer and Information Sciences, College of Science and Technology, Covenant University Ota, Ogun State, Nigeria, under the supervision of Professor Charles Korede Ayo and Dr. Olawande Daramola. We have examined and found the work acceptable for the award of a degree Doctor of Philosophy in Computer Science.

PROFESSOR CHARLES KOREDE AYO	_____
(Supervisor)	Signature and Date
DR. OLAWANDE DARAMOLA	_____
(Co-Supervisor)	Signature and Date
DR. OLUFUNKE OLADIPUPO	_____
(Head of Department)	Signature and Date
PROFESSOR CHARLES UWADIA	_____
(External Examiner)	Signature and Date
PROFESSOR SAMUEL WARA	_____
(Dean, School of Postgraduate Studies)	Signature and Date

DEDICATION

This work is dedicated to God Almighty who has made me a pen in the hand of the ready writer, ordering my steps and guiding my way through the journey of life and ensuring that everything works together and is fitting into a plan for good for me at all times.

It is also dedicated to Him only.

ACKNOWLEDGEMENTS

At the end of every seven years, you shall grant a release.

...because the LORD's release has been proclaimed.

I am deeply indebted to God my Father, the author and finisher of this work for giving me wisdom and understanding needed to complete this research. His grace saw me through my doctoral studies.

My deep and sincere gratitude also goes to the Chancellor, Dr David Oyedepo and the members of the Board of Reagents of Covenant University Ota for the vision and mission of the University.

Also special thanks to the management and staff of the University: the Vice Chancellor, Prof. AAA Atayero, the acting Registrar, Mrs Mary Aboyade, the Deans of the Colleges and the various Heads of Department for their commitment to the pursuit of the vision of Covenant University.

I would also like to use this medium to express my sincere gratitude to the Head of Department, Dr Olufunke Oladipupo for her motivation towards completing this project, my supervisor Professor. C.K Ayo for his fatherly role and mentorship in my career and finally my co-supervisor, Dr Olawande Daramola, who inculcated in me a sense of standard and quality in the course of this project, his critical views, observations, and suggestions were the major driving force for the successful completion of this work.

My appreciation also goes to the entire staff of the Department of Computer and Information Sciences for their continuous encouragement and support in times of need, also for their critiques and contributions during presentations at the departmental and cluster levels. God will reward you all.

I remain thankful to the management of Covenant University Ota, for availing me the opportunity to be a beneficiary of the Young Academics Training Programme (YATRAP) initiative. May God continue to take us forward to greater heights in Covenant University.

To my friends (Sona Odafen and Joy Lamptey) who contributed to the content and editing of this work, thank you for being there for me. To my wife (Shalom Obi), thank you for being by my side all through this work. To my project students at the undergraduate programme especially Izuchukuwu Alaine, I want to appreciate you all for your immense contributions to this work. To my other colleagues in the YATRAP program thank you for being a big family to me here.

I would like to acknowledge and appreciate my parents Mr Christopher and Mrs Ada Emebo, for investing their lives and resources to providing me quality moral, formal education and spiritual upbringing, also for all their parental support, encouragement and belief in me. To my siblings (Uchenna, Ada, and Ifeoma), I thank you all for your believing in me as well and for the prayers rendered ceaselessly in seeing to it that this work is completed.

Not forgetting the Fulbright Scholarship programme that gave me the unique opportunity to go for my fieldwork in the United State of America (U.S.A) where I had the opportunity of being hosted by Montclair State University under the supervision of Dr Aparna Vardea. It was indeed a lifetime opportunity to interact with the software industry in getting involved in the evaluation of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABR	Analogy-Based Reasoning
API	Application Programming Interface
AI	Artificial Intelligence
ADO	Application Domain Ontology
BFO	Basic Formal Ontology
CBR	Case-Based Reasoning
CMS	Course Management System
NLP	Natural Language Processing
IDE	Integrated Development Environment
IMR	Implicit Requirements
POS	Part of Speech
PROMIRAR	PROduct for Managing Implicit Requirement using Analogy-based Reasoning
RE	Requirements Engineering
RT	Requirements Traceability
SRS	Software Requirements Specification
SE	Software Engineering
UML	Universal Modelling Language
WSD	Word Sense Disambiguation

ABSTRACT

The ability of a system to meet the stated requirements affects the success and overall usability of the system. The presence of implicit requirements create difficulties in fulfilling the desires and needs of the stakeholders during software development. Identification of implicit requirements is essential to the functionality of the software as implicit requirements are equally as important as explicit requirements. Although different researchers and practitioners have identified the importance of implicit requirements for the overall successful outcome of software development, there is a need to correlate these theoretical assumptions about implicit requirements with the state of practice and also create a framework which can effectively identify and manage implicit requirements within a software organisation. This thesis is a two-part research. It involved an empirical investigation of the perception and handling of implicit requirements in small and medium-sized software organisations and the presentation of a process framework to identify and manage implicit requirements during software development process. The empirical investigation was conducted using a survey, which was conducted through a web-based questionnaire, where 56 participants from 23 countries participated. The study found that critical organisational factors such as number of years in the business of an organisation, the years of experience of an organisation in requirements engineering, and size of software development team have a positive correlation with the perception and handling of implicit requirements within an organisation. Further analysis showed that a significant number of practitioners believe that additional means can complement the use of experience such as tool support in managing implicit requirements. Hence, the relevance of the second part of this research, which presents an approach for identification and management of implicit requirements using analogy-based reasoning, ontology, and natural language processing. The approach is supported by a prototype tool, which was assessed by conducting a performance evaluation of the tool with industry experts as well as with three other existing tools. From the performance evaluation result, the prototype tool had a mean recall value of 83.20% and a mean precision of 86.16% showing that the tool is efficient and fit for practical use. Also from the comparative analysis done, firstly, it was observed that the lexical ambiguity and structural ambiguity analysis of the prototype tool performed better than the first tool in terms of recall and F-Score but were almost at par in terms of precision. Secondly, when the lexical analysis of the prototype tool was compared with the second tool, they both performed at par across all metrics. Finally, when the vagueness analysis of the prototype tool was compared with the third tool, it was observed that the prototype tool performed better across all metrics. An industrial evaluation of the process framework with two requirements management tool by two experts each from two companies was conducted, which further revealed that the prototype tool integrates well with organisational requirements engineering processes. Recommendations were made towards improving the domain ontology for enhanced implicit requirements identification. In conclusion, the ability to discover unknown and un-elicited requirements will mitigate many risks that can adversely affect system architecture design and project cost.

Keywords: analogy-based reasoning, implicit requirements engineering, ontology.

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND INFORMATION

Every software system has a pre-determined purpose and the success of such a system is dependent on its ability to achieve this purpose. The purpose and functions of a software system, however, depends on the needs of the stakeholders of which the system is considered a failure if it is unable to adequately meet these needs. Hence, all the requirements of the system must be met.

Requirements engineering is a core activity in software development that refers to a process that covers key foundational activities such as discovering, documenting and maintaining the requirements for achieving the predefined purpose of a system (Kotonya and Sommerville, 1998). It connects the needs of the stakeholders and the functions of the software system, thereby creating optimum satisfaction for the users of the system.

Requirements engineering is a systemic process which begins with requirements elicitation and involves the identification of requirements from different stakeholders. At this stage, different types of requirements are identified and they can be broadly classified into two, namely: explicit (this includes both functional and non-functional requirements) and implicit requirements. Explicit requirements refer to the well-defined and clearly stated requirements that a system should execute while implicit requirements (IMR) are the hidden or assumed requirements that a system is expected to fulfil even though not explicitly elicited during requirements gathering (Daramola *et al*., 2012; Pittke *et al*., 2015).

As an example, the scenario of creating a portal system for a publishing company that was extracted from Jha (2009) was considered. While capturing the requirements of the system, an important requirement was omitted. The following requirements were documented.

i. **Explicit Requirement 1**: A Publisher should be able to create an article, send it for approval and finally publish the article on the portal.
ii. **Explicit Requirement 2**: A Portal user should be able to search articles published on the Portal.

The objective was to develop a system that will cater to both requirements so that the publisher could create and publish an article and the end user could search the articles using the search functionality. However, when the system was developed and deployed, end user discovered that the search result displayed articles, which were published, under approval and in draft state. This was unsatisfactory for the end-user of the system. This outcome was as a result of an uncaptured (unspoken) implicit requirement, which was never mentioned, and thus was not catered for prior to the development of the system.

i. **Implicit Requirement**: The articles in draft state and under approval should not be displayed in the search result.

According to Ahamed (2010), the quality of any software system is dependent on its conformance to both explicit and implicit requirements. Hence, the quality of software cannot be adjudged good and guaranteed to meet customers' satisfaction if only explicit requirements are satisfied while the implied ones are ignored (Drysdale, 2007).

Even though IMR is essential to the successful function of any software, its undefined nature serves as a challenge causing it to receive less attention in practice. This differs from explicit requirements which, as opposed to IMR, are clearly defined (Grehag, 2001; Singer *et al*., 2009). So far, IMR is handled by requirements engineers who use their initiative and experience to address the challenges that the absence of such requirements pose to the overall purpose and functions of the system (Jha, 2009; Parameswaran, 2011).

A number of reasons can lead to the emergence of IMR, some of which are: i) there is lack of implicit shared understanding among all the stakeholders in a project about the quality of requirements of a system (Glinz, 2014); ii) a software organization develops a product in a new domain or subcontracts the software to an external organization with a different operational background via outsourcing (Deshpande and Richardson, 2009) and iii) there

is a knowledge gap between developers and shareholders due to existence of implicit knowledge. Failure to make IMR explicit in such instances could lead to serious problems (Polyani, 1983).

According to Polyani (1983), implicit knowledge means "*knowing more than you can tell"*. If a stakeholder offering information on a system needs to know more than can be told, then not all that is known will be told. This missing implicit knowledge can become a basis for some IMR, which the stakeholder expects the system to fulfil when developed (Gacitua *et al*., 2009). Based on the reasons outlined above, it is sufficient to say that the inability to effectively manage IMR can lead to poor software quality, which in essence creates a shortfall between stakeholders' needs and the functionality of the system leading to customers' dissatisfaction with the software.

In addition, managing IMR pose some challenges during software development because i) what is classified as IMR at a certain time, can become obsolete over time because of technology advancement and new innovations; ii) IMR can lead to budget deficit if not well managed. For instance, if IMR are not discovered on time, it could be more costly to integrate such requirements into design or implementation stages, which will swell the budget for the project; iii) IMR that pertain to software architectural concerns such as scalability, maintainability, and usability if not properly understood and addressed on time carry significant risks that can adversely affect the success of a project (Daramola *et al*., 2012; Douglass, 2009; Singer *et al*., 2009).

According to Daramola *et al.* (2012), these challenges can be partially tackled using good requirements elicitation methods or through inclusive software development paradigms such as agile approaches. It is, however, important to note that there are certain situations where these options are not the preferred choice as these approaches are not known to possess the inherent capability for handling IMR.

In addition, the advent of requirement management tools such as DOORS, Caliber-RM, RDD-100, RequisitePro, and icCONCEPT-RTM, amongst others (Larsson and Steen, 2008), and requirement analysis techniques like the KANO model (Kano *et al*., 1984; Xu *et al*., 2007), have failed to directly address issues of IMR. Closely related to the intention

of this research is the MaTREx project (Gacitua *et al*., 2009). In MaTREx, the focus of the authors was on evolving tools and techniques to improve the management of information on requirements via automatic trace recovery; ascertaining the existence of tacit knowledge by tracing of unprovenanced requirements and presuppositions; and uncovering in requirements documents, the presence of nocuous ambiguity that would result in potential misinterpretation. However, the focus of this thesis is to use analogy based-reasoning approach for effective discovery and explicit documentation of IMR, which would help to manage IMR in a way that improves the overall success of software development processes.

In view of this, this research proposes a tool support framework that can be integrated into an organisation's Requirement Engineering (RE) procedures in order to manage IMR. This is to improve the efficiency of the RE process, and eventually the whole software development task.

1.2 STATEMENT OF THE PROBLEM

Implicit and explicit requirements are both crucial to the success of a software system (Drysdale, 2007; Grehag, 2001; Leffingwell and Widrig, 2000; Pittke *et al*., 2015). So far, very little attention has been accorded IMR in the literature as opposed to loads of attention lavished on explicit requirement as a subject matter (Daramola *et al*., 2012; Kotonya and Sommerville, 1998). The observations and general opinions of practitioners suggest that IMR poses a lot of challenges for software developers, hence the need to efficiently and effectively manage them. Although there is a general perception of the importance of IMR during software development, there is yet a lack of empirically proven evidence through research studies that have assessed the impact of IMR on the success or failure of software development projects.

Therefore, the research questions investigated in this work are:

i. What is the impact of IMR on system development in practice?

ii. How can IMR be efficiently managed within an organisational context to promote successful RE during software development?

1.3 AIM AND OBJECTIVES OF THE STUDY

This research work aims to evolve a process framework for managing IMR within an organisation.

To achieve this aim, the objectives of this work are to:

i. empirically investigate the impact of IMR on success or otherwise of software development in practice;

ii. design a process framework that both discovers and manages IMR in a systematic way;

iii. provide a prototype tool support for the process framework for managing IMR; and

iv. Evaluate the process framework and prototype tool using industrial case studies and controlled experiments.

1.4 RESEARCH METHODOLOGY

In order to fulfil the goal of this research work, the primary research methods used are empirical survey and design science research methods.

This includes literature survey, empirical survey, tool prototyping, case study and controlled experimentation. To achieve the aim and objectives of this work, the research framework adopted for this research is premised on the framework presented in Hevner *et al.* (2004) and Figure 1.1 shows the flowchart of the adopted research process. The following paragraphs highlight the activities engaged in this research.

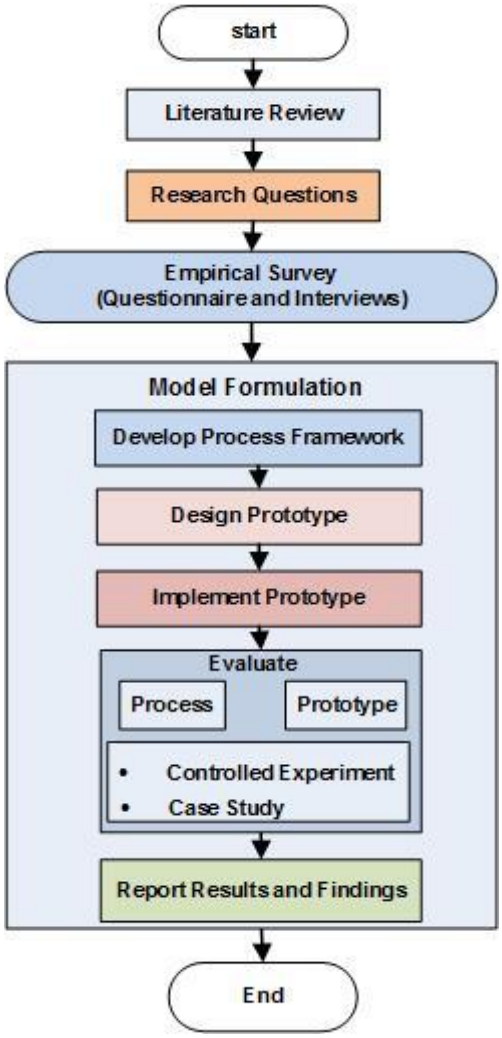


Figure 1.1: Flowchart of the adopted Research Process
Source: (Hevner *et al*., 2004)

a. **Literature Survey:** The literature survey includes the review of relevant literature that contains valuable information, which could be useful to this research work. This method involves in-depth research of the essential concepts (requirements engineering, IMR, requirements management amongst others), which form the foundational part of this research work. Also, past research works, which are directly and indirectly related are reviewed. Materials reviewed include published journals, peer-reviewed conference papers, and technical reports.

b. **Empirical Survey:** An empirical survey aimed at investigating the impact of IMR on the software development process in practice was carried out. Although it has been speculated in the literature that poor management of IMR leads to poor quality of

software product, this has not been widely supported by empirical evidence. Hence, a survey was carried out involving RE experts in the academic and professional global online communities (e.g. SEWORLD, AISWORLD, Yahoo requirements engineering group, etc.). The empirical data was gathered by using an online questionnaire.

c. Model Formulation: From the result of the survey of the literature, this thesis proposed the Analogy-Based Reasoning (ABR) approach in order to manage IMR. The approach integrates two other core technologies, which are Natural Language Processing (NLP) and Ontology. These three technologies were then integrated in a complementary way in order to formulate an architectural framework for managing IMR.

d. **Proof of Concept Implementation:** A prototype tool (PROMIRAR) was developed to support the process of managing IMR. The tool is lightweight and developed with the Eclipse IDE that can be integrated with other Eclipse-based requirements management tools or other open source requirements management tools.

e. **Evaluation:** The prototype tool and the process framework were evaluated using industrial case studies and controlled experiments. This entails a field assessment of the tool by industrial experts and controlled experiments using industrial experts, faculty and graduate students of software engineering program. Thereafter, the result of the evaluation experiments was analysed in order to establish a basis for the generalisation of the results.

1.5 SIGNIFICANCE OF THE STUDY

This research work has significance to the RE community and the software development industry in general. The importance of this study includes the following:

i. This study identifies some gaps in existing requirements management literature and proposes a process framework that enables the effective management of implicit requirements during RE phase of software development.

ii. The developed systematic tool support framework when integrated into an organisation's RE procedure for managing IMR. This will promote the efficiency

of the RE processes in software development organisations, thereby reducing the occurrence of software budget overrun by software firms.

iii. It will further enhance the quality of the software product delivered thereby eliminating instances of poorly developed software products.

iv. It will bring about greater user satisfaction in developed software products.

1.6 SCOPE OF THE STUDY

The main focus of this work is to investigate how implicit requirements (IMR) can be effectively and efficiently managed within small and medium-sized software organisations that embark on projects that solve problems within their immediate environment. These organisations are more in number and easily accessible. However, little emphasis was placed on large-sized organisations. Also, this research is focussed on companies that are developing information systems and also lightweight embedded systems. Organisations that are involved in the development of more complex systems and cyber-physical systems were not considered.

1.7 ORGANISATION OF THE THESIS

Chapter One of this thesis presents a general introduction to the study, the statement of the problem, the aim and specific objectives of the research, the methodology used, scope of the study and the significance of the study.

Chapter Two undertakes a critical review of the literature on the key issues and activities in Requirements Engineering (RE) with a focus on implicit requirements. The chapter presents a critical review of implicit factors in RE touching on the sources of implicitness and an overview of core technologies and concepts that are essential to the handling of implicitness in RE. The chapter also reviews different works that attempt to convey the importance of IMR in RE. The Chapter concludes finally with a brief summary of findings.

Chapter Three describes the research methodology adopted by this thesis. The first part discusses the design and implementation of the empirical survey, while the second part describes the architecture of the process framework for managing implicit requirements.

Chapter Four presents a description of the design and implementation of the support tool-PROMIRAR for identification and managing IMR.

Chapter Five handles the evaluation procedure of the approach as well as the tool developed and it concludes with a discussion of the threats to the validity of the industrial case evaluation.

Chapter Six is the last chapter that contains the summary, conclusion and a discussion of the future research outlook of this thesis.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter provides a review of literature in order to properly situate the context of this research. It presents theoretical background on important aspects such as requirements engineering, and the core technologies that have been used in this research – Ontology, Natural language processing (NLP), Analogy-Based Reasoning (ABR). It closed with a review of sources of implicitness in requirements engineering and related work on implicit requirements.

2.2 OVERVIEW OF REQUIREMENTS ENGINEERING (RE)

Before delving into the details of software requirements engineering, it is essential that the words making up the concept of requirements engineering be properly defined and explained to ease understanding. Thus, there is need to understand what *requirements* and *engineering* are, as used in this study. Kotonya and Sommerville (1998) gives the definition of *requirements* as a specification of what should be implemented or a constraint of some kind on a system software; usually defined during the early stages of software development. He further sees requirements in the form of a user-level facility description, a detailed specification of expected system behaviour, a general system property, a specific constraint on the system, information on how to carry out some computation and a constraint on the development of a system software among others. The use of the term *engineering* implies that systematic and repeatable techniques be used to ensure that system requirements are complete, concise and relevant among other things. This can be seen resonating all through the requirements engineering process from the elicitation of requirements to the modelling and analysing of requirements, negotiation, validation and verification and finally the management of change.

Different researchers have given their definitions of requirements engineering. According to Kotonya and Sommerville (1998), requirements engineering refers to that aspect of software engineering that comprises all activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system. It is considered to be the initial phase of the lifecycle of software development that deals with the identification, documentation and management of requirements (Odeh, 2009). It usually involves lengthy deliberations among stakeholders to create a platform for them to conclude and complete accurate and unambiguous list of software requirements. Zave (1997), defined requirements engineering as "*the branch of software engineering concerned with real world goals for functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour and to their evolution over time and across software families*". The definition goes further to reiterate requirements engineering not only as a process of software development but also as a core engineering activity. The definition reiterates requirements engineering (RE) as an important part of an engineering process pointing to the fact that RE is concerned with anchoring development activities to a real-world problem in a way that the appropriateness and cost-effectiveness of the solution can be analysed (Nuseibeh and Easterbrook, 2000). The requirements engineering process is concerned with the identification, modelling and verification of the functionality of a software system. It also puts into cognizance the context within which it will be developed and operated. Pohl (1997) itemised four (4) main tasks of requirements engineering as requirements elicitation, negotiation, specification and validation or verification. These are simply the basic activities that make up the process of requirements engineering. Requirements engineering has however been considered a misnomer with a school of thought arguing against its being referred to as an engineering process while the other faction sees it as an integral part of software engineering (Pohl 1997).

Over the years different problem areas have been identified in software development, however, meeting the requirements of the customer or user of the software is one which is always emphasised. The principal problem areas in software development and production as put forward by the European Software Process Improvement Training Initiative, 1996,

are the specification, documentation and management of requirements which is a sum total of what requirements engineering (RE) is about. It was further discovered that difficulties with requirements in embedded systems are the root-cause of safety-related software errors that have persisted until integration and system testing.

It is of utmost importance that requirements are clearly specified and documented; free from any form of ambiguities, inaccuracies and inconsistencies. If this is not the case, then complications could arise in the software development process, which may result in the system being delivered late and costing much more than originally expected. It could also result in the system becoming unreliable in use with regular system errors and crashes disrupting normal operation or better still in the dissatisfaction of customers and end-users of the software with the system, making the identification and handling of such issues with requirements as is usually the case with implicit requirements very necessary. Requirements engineering is however not an easy task. It comes with its own baggage of difficulties, which could arise from the system itself or from external sources such as the stakeholders of a particular software system and the environment within which a software system is to operate. Change is a major difficulty that requirements engineering is saddled with. Different things that the requirements engineer works with are subject to change. From the environments within which a software system operates to the goals and priorities of stakeholders and even their needs and the specifications of the system software.

The practice of requirements engineering in the 21st century has been faced with several challenges with some of these challenges being E-commerce and globalisation, accelerated system development and off-the-shelf systems among others.

The ideas that brought such drastic changes to the field of Requirements Engineering (RE) are that not only should the process of modelling and analyzing requirements be done using contextualized enquiry techniques as opposed to techniques isolated from the organizational and social context in which the software system would have to operate but that RE should focus on modelling indicative and optative properties of the environment, not just the functionality of the new system (Zave and Jackson, 1997) and that the requirements engineer has to take seriously the need to analyze and resolve conflicting

requirements to support stakeholders' negotiation and also to reason with models that contain inconsistencies (Ghezzi and Nuseibeh, 1998).

In their opinion, RE is a multidisciplinary, human-centered process whose tools or techniques draw upon a variety of disciplines making it a necessity for the requirements engineer to master skills in a variety of disciplines including cognitive psychology, anthropology, sociology and linguistics amongst others, which helps the requirements engineer to develop a software system that would enhance and not hinder human activities.

Requirements engineering can, therefore, be seen as the process of discovering the purpose for which a software system is to be developed by identifying stakeholders and their needs and documenting same in a form that is responsive to analysis, communication and subsequent implementation. This underlies a major component of system development processes. Thus, if attention is given to requirements, the probability of the system developing problems in future is reduced.

2.2.1 Key Issues in Requirements Engineering

Requirements engineering as a field has several issues arising as it only became popular in the 1990s. It evolved as a field in its own right following several important and radical shifts in its understanding.

The first issue arising in requirements engineering is the fact that it is a relatively new field in which several groundbreaking discoveries are still being made. In fact requirements engineering could not attain the independence of being a field in its own right due to several orthodox views that made up its components. Thus, the need for the emergence of new ideas that clearly defined what requirements engineering should entail.

Secondly, requirements engineering is considered a misnomer amongst scholars due to it being considered a core engineering activity (Zave, 1997). However, from the definitions given of engineering by several textbooks, there is no doubt that requirements engineering is indeed an engineering activity as it seeks to provide cost-effective solutions to real-world

problems in such a way that the cost-effectiveness and appropriateness of the solution can be analysed.

2.2.2 Activities in Requirements Engineering

The areas of requirements engineering, which is a sum total of all the activities that make-up the process of RE are eliciting requirements, analysing and modelling requirements, communicating requirements, agreeing on requirements and evolving requirements (Kaur and Singh, 2010).

I. Elicitation of Requirements in RE

The elicitation of requirements is usually the first step in the RE process and it is worthy of note that it does not imply that requirements are out there to collect only by asking the right questions but that the information gotten from stakeholders must be interpreted, analysed, modelled and validated and more often implied requirements elicited before the requirements engineer can be confident he has what he needs to develop a particular software system.

An important goal of requirements elicitation is to find out the purpose for which a software system is being developed by identifying the problem such a system is to solve. This helps the requirements engineer to identify system boundaries, which would determine where the final delivered system would fit into the current operational environment (Nuseibeh and Easterbrook, 2000).

There are several techniques that can be employed in the requirements elicitation process in RE. The type of elicitation technique favoured by the requirements engineer, however, depends not only on the time and resources available to him or her but also on the type of information to be elicited. Among the techniques of requirements elicitation is the traditional techniques, which include the use of questionnaires and surveys, interview and the analysis of existing documentation such as organisational charts, process models or standards, and user or manuals of existing systems.

The group elicitation technique is another elicitation technique that includes brainstorming and focus groups and the use of consensus-building workshops with unbiased facilitators with the aim of exploiting team dynamics to elicit a richer understanding of needs, which would, in essence, serve as requirements. Prototyping is used for elicitation when there is a great deal of uncertainty about requirements or where early feedback is required from stakeholders. This technique can be combined with other techniques to achieve optimum results.

Model-driven techniques provide a specific type of model for the type of information to be gathered and use this model to drive the elicitation process while cognitive techniques include a series of techniques originally designed for knowledge acquisition for knowledge-based systems (Shaw and Gaines, 1996) and it includes protocol analysis where an expert thinks aloud while performing a task in order to afford the observer an insight into the cognitive process used to perform the task.

Finally, contextual techniques involve participant observation and arose in the 1990s as an alternative to the traditional and cognitive techniques of requirements elicitation. From the plethora of techniques available, the requirements engineer would have to select the appropriate technique or techniques most suitable for the process of requirements elicitation at hand making technique-selection guidance more appropriate than rigid methods in the process of requirements elicitation (Maiden and Rugg, 1996).

II. Modelling and Analysing Requirements

The modelling and analysis of requirements are the next activity undertaken in the RE process after requirements elicitation. Modelling can be said to be the construction of abstract descriptions that are agreeable to interpretation in the sense that the models created are used to represent a whole range of products of the requirements engineering process (Nuseibeh and Easterbrook, 2000). The aspect of analysis in the second stage of the RE process refers to the generation of useful information from the models produced or created.

This goes to say that after the requirements engineer has elicited necessary requirements using the elicitation technique that best suits the particular situation, he proceeds to create or produce models to serve as drivers for further information gathering while at the same time uses a particular approach to analysis to be able to generate useful information from the models thereby enabling it to achieve the purpose for which it was produced.

There are several modelling techniques that can be utilised as a tool through which models can be made to elicit the necessary information needed. Some of these techniques include enterprise modelling, which entails having an understanding of an organisation's structure; the business rules that affect its operation; the goals, tasks and responsibilities of its constituent members and the data that it needs to generate and manipulate. What this means is that enterprise modelling captures the purpose of a system by describing the behaviour of the organisation in which that system will operate (Loucopoulos and Kavakli, 1995).

Data modelling is another modelling technique that puts into cognisance the fact that large computer-based systems, information systems more specifically, generate large volumes of information that must be understood, manipulated and managed. Thus, it creates the opportunity for the issue of managing large volumes of information when dealing with large computer-based systems in requirements engineering to be addressed.

Behavioural modelling and domain modelling are also techniques used in modelling and analysing requirements in RE. While the former places emphasis on the dynamic or functional behaviour of stakeholders and systems, the later focuses on developing domain descriptions, which would provide an abstract description of the world in which an envisioned system will operate. Modelling requirements help the requirements engineer in the easy analysis of the requirements as it puts techniques such as requirements animation, automated reasoning, analogical and case-based reasoning and knowledge-based critiquing among others at the disposal of the requirements engineer with which he is able to analyse the models produced to generate information.

III. Communicating Requirements

Requirements engineering goes far beyond discovering and specifying requirements. It also puts the fact that for any system software to achieve the intent for which it is developed, it must be communicated to the different stakeholders in such a way that they are not only able to understand it and the process behind its creation and implementation but are also able to interact with the system right from the point where requirements are elicited to the stage where the system is implemented thus enabling a cross-breed of ideas that would further increase the functionality of the system.

It would, therefore, not be out of place to opine that the requirements engineering process goes beyond the mere identification and specification of requirements to being able to communicate effectively the requirements that have been elicited to the participating parties, thus, creating a shared meaning upon which several other requirements though not explicitly stated could be brought to the fore and those already available analysed easily, written and revalidated.

Communicating requirements, however, goes beyond just eliciting requirements to managing these requirements. What this means is that the requirements engineer is just not only able to write out requirements elicited but also able to so in such a way that it is readable and traceable by many in order to manage their evolution over time making it a core scientific process.

Requirements traceability (RT) according to (Gotel and Finkelstein, 1994) is the ability to describe and follow the life of requirements from its origin through its deployment and use up until it is converted to a software system. It is a crucial factor in requirements engineering that determines how easy it is to read, navigate, query and change requirements documentation (Nuseibeh and Easterbrook, 2000). Providing the elements of requirements traceability in requirements documentation helps the documentation to gain a level of integrity and completeness that would be useful in managing change in the process of requirements engineering as is always the case.

IV. Verifying and Validating Requirements

It is of utmost importance that an agreement is created and maintained between stakeholders irrespective of their divergent needs, opinions and goals during the requirements engineering process. This, therefore, makes it absolutely necessary to explicitly describe requirements as it not only makes validating requirements possible but also creates a platform through which conflicts between stakeholders can be resolved.

Validating requirements, such as validating scientific knowledge, is not without its difficulties. The first difficulty encountered in creating validation in requirements engineering is philosophical in nature as it concerns what truth is and what could be considered knowable. It is for this reason that Popper (2009), opines that scientific theories can never be proved through observations, only refuted.

The second difficulty that arises when creating validation in requirements engineering is social. This difficulty concerns how agreement in requirements engineering can be reached in spite of the conflicting needs and goals of the diverse stakeholders involved in the process. It is based on the problem of disagreement between stakeholders, which can be managed by a process known as requirements negotiation. Requirements negotiation, therefore, attempts to resolve the conflict among stakeholders while still ensuring that the satisfaction of each stakeholder's goal is not undermined in any way (Ghezzi and Nuseibeh, 1998).

V. Managing Change in Requirements Engineering

The management of change is of utmost importance in requirements engineering. It is in fact considered a fundamental activity in requirements engineering. This is because successful software systems are dynamic in nature, evolving alongside the environments in which they operate (Arnold and Bohner, 1996).

Requirements evolve due to several reasons, which include but are not limited to adding or deleting requirements or fixing errors. These usually occur in response to the changing needs and goals of stakeholders; being humans they are dynamic and so are their needs and

goals. The need to manage inconsistencies in requirements documentation, which could have arisen as a result of mistakes or conflict between requirements also creates evolving requirements in requirements engineering.

It is, however, important to note that each proposed change has to be evaluated in terms of existing requirements and architecture of existing software system so that acceptable trade-off between the cost and benefit of making a change can be achieved. The process of identifying core requirements in order to develop a successful software system that is stable in the presence of change and flexible enough to be customised and adapted to changing requirements is simply what evolving requirements in the RE process are about.

2.3 CORE TECHNOLOGIES AND CONCEPTS

In this section, an overview of the core technologies and concepts that relate to the context of this research is presented. In tackling the challenges of IMR, this research engaged the use of a combination of artificial intelligence technologies, namely Analogy-Based Reasoning (ABR), Ontology, and Natural Language Processing (NLP). These approaches play the following role in achieving the research objectives: ABR - enables cross-domain reuse of previous requirements specifications in the discovery of new IMR; Ontology – enables the representation of relevant domain knowledge that is crucial for managing IMR in specific domains and NLP – enables the automated analysis of requirements, since requirements are mostly written using natural language texts. Hence, the combination of these approaches has the potential to address issues of managing IMR during RE.

2.3.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a multidisciplinary subject and it is generally defined as the processing of human language. It deals with a representation of human knowledge, which has the potential of having ambiguities without the knowledge of the speaker and the receiver. Therefore, this can be tedious if natural language is processed manually, making it either automatic or semi-automatic. Natural language cuts across different fields. In Linguistics, NLP focuses on formal, structural models of language and the discovery of language universals. In Psychology, it gives insight into the human

cognitive process with the aim of modelling the language to a compressible form. In the field of Artificial Intelligence (AI), NLP focuses on the interaction between humans and computers. It focuses on developing internal representations of data and efficient processing of these structures. Different researchers have different definitions for the term NLP. (Chopra *et al*., 2013), defined NLP as a subfield of Artificial Intelligence and Linguistics, devoted to making computers understand the statements or words written in human languages. According to Liddy (2001), NLP is a theoretically motivated range of computational techniques for analysing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications. Mooney (2014), defines NLP as a branch of computer science focused on developing systems that allow computers to communicate with people using everyday language. A consistent fact with these definitions is the interaction between computers and humans. This means there is an input of natural language which is converted to machine language (for computer understanding) and the output is released as natural language. At the stage of the input, the computer/machinery has natural language understanding, which is the task of reasoning and understanding where the input is natural language while the output stage refers to the generation of natural language, which can be in the form of text or other forms.

NLP has evolved over many decades. A notable time in the field of NLP is in 1950. Alan Turing released an article titled *Computing Machinery and Intelligence*. The article proposed what is now referred to as the Turing Test. Turing test focused on the replication of human intelligence behaviour in machines. A human evaluator that can differentiate between the machine and human participants was proposed to serve as a judge of natural language conversations (text only) between a human and a machine that is designed to generate human-like responses. Since then there have been other notable and successful NLP systems. For example ELIZA, SHRUDLU are successful NLP systems of the 60's which were based on block worlds, In the 70's the introduction of ontologies into NLP system gave birth to systems such as MARGIE, SAM Talespin and many other NLP systems.

Different researchers have given different interpretations to the various stages of NLP. Liddy (2001), classified the natural language process into the following levels: Phonology, Morphology, Lexical, Syntactic, Semantic, Discourse and Pragmatic.

As shown below in Figure 2.1 is the diagrammatical representation of the stages in NLP.

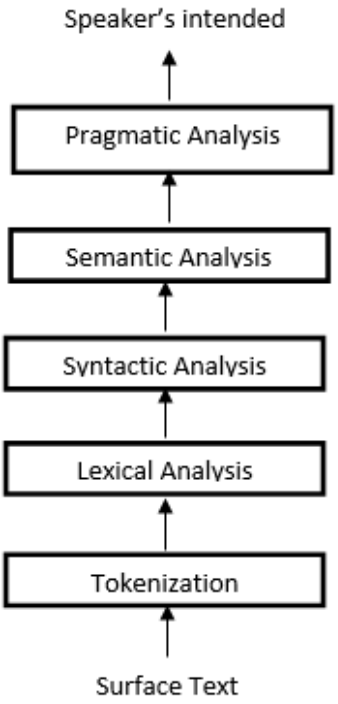


Figure 2.1: Stages in Natural Language Processing
Source: (Liddy, 2001)

I. Text Preprocessing

Another word for text preprocessing is Tokenization (as shown in Figure 2.1) or Text Normalisation. This is an essential part of the NLP system, which reformats the original text into meaningful units that contain important linguistic features before performing subsequent text mining strategies. Palmer (2010) defined text preprocessing as the task of converting a raw text file, essentially a sequence of digital bits, into a well-defined sequence of linguistically meaningful units. The units are at the lowest level characters representing the individual graphemes in a language's written system, words consisting of one or more characters, and sentences consisting of one or more words. It is an essential

part of any NLP system, since the characters, words, and sentences identified at this stage are the fundamental units passed to all further processing stages, from analysis and tagging components, such as morphological analysers and part-of-speech taggers, through applications, such as information retrieval and machine translation systems. This process includes activities such as segmentations, punctuations, part of speech and sentence parsing. Text preprocessing can be classified into two: Document Triage and Text segmentation. Document Triage can be defined as the process of converting a digital file into a well-defined text document and it includes activities such as character encoding identification and text sectioning. Wei *et al.* (2013) defined Document triage as the stage of converting a binary file into a well-defined text document. This is an easily achievable stage with the right available software tools. However, it is also crucial to the results as errors at this stage can make the results incomprehensible. Text Segmentation in simple terms refers to the breakdown/ division of the text into meaningful units such as sentences, words, topics, amongst others. There is different type's text segmentation. One of such is Morphology. This refers to how the words are broken down into Morphemes (smallest units of meaning). Morphemes meanings remain the same across words, therefore, an NLP system can recognise the meaning conveyed by each morpheme in order to gain and represent meaning (Liddy, 2001).

II. Lexical Analysis

Lexical Analysis refers to the identification and analysis of the structure of words. This involves the conversion of words into a sequence of tokens. The lexical analysis involves the division of texts into paragraphs, sentences, and words. The aim of lexical analysis in natural language processing is to connect each word with its corresponding label in a dictionary. However, many words have multiple meanings (depending on the context in which it is used), making it almost impossible to choose the correct meaning of the word considering only the highlighted word in its context. This means that in an instance, a word in a grammatically valid sentence can be replaced by another of the same grammatical class, maintaining the validity of the sentence. Within the same class of words, there are groups of rules that characterise the behaviour of a subset of words from one language. A

morphological analyser is a key to understanding a sentence, as to form a coherent structure, it is necessary to understand the meaning of most of his words forming. It identifies words or phrases in one sentence alone, marking each with a token symbol. This process is aided by delimiters (punctuation and blanks), that is recognised in the stage of preprocessing. The tokens identified are classified according to their use (grammatical class). The primary aim of the lexical analysis phase is the segmentation the input stream of characters into tokens, simply grouping the characters into pieces and categorising them (Liddy, 2001).

III. Syntactic Analysis

Syntactic analysis is also known as parsing. It is derived from the Latin word Pars (*orationis*), which means parts of speech. This stage in natural language processing has to do with the analysis of a string of symbols, either in natural language, conforming to the rules of grammar. Ljunglof and Wiren (2010), defined syntactic analysis as the analysing of a string of words (typically a sentence) to determine its structural description according to a formal grammar. It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship between the words. A grammar is used to determine what sentences are legal and it is applied using a parsing algorithm. This analytical process results in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information. The parse tree breaks down the sentence into structured parts so that the computer can easily understand and process it. The major purpose of the syntactic analysis is to analyse the syntactic structure of the program and its components and to check these for errors. The string of words used as input are outputs of lexical analysis and tokenization respectively.

There are different types of grammar, which have been developed by different researchers. A grammar consists of one or more variables that represent classes of strings. There are rules that say how the strings in each class are constructed. The construction can make use of the culmination of symbols of the alphabet and strings that are already known to be in one of the classes or make use of them separately. One of such is the Context Free Grammar (CFG). It was introduced by Chomsky (1956). It is used to describe the syntactic structures

of a programming language. It describes what elementary constructs there are and how composite constructs can be built from other constructs. A CFG can be defined as a tuple G = (V, T, P, S)

where

i. V is the (finite) set of variables (or non-terminals or syntactic categories). Each variable represents a language or a set of strings.
ii. T is a finite set of terminals. This refers to the symbols that form the strings of the language being defined.
iii. P is a set of production rules that represent the recursive definition of the language.
iv. S is the start symbol that represents the language being defined.

Other variables represent auxiliary classes of strings that are used to help define the language of the start symbol. CFG is the most influential grammar formalism for describing language syntax. It is the most simple and most generally adopted. Also, most formalisms are derived or related to the Context Free Grammar.

IV. Semantic Analysis

A classic view of Semantic Analysis was given by Poesio (2000), which states that the ultimate goal, for humans as well as natural language processing systems, is to understand the utterance, which, depending on the circumstances, may mean incorporating information provided by the utterance into one's own knowledge base or, more, in general, performing some action in response to it. 'Understanding' an utterance is a complex process, which depends on the results of parsing, as well as on lexical information, context, and commonsense reasoning (Poesio, 2000).

In general, linguistics semantic analysis refers to analysing the meanings of words, fixed expressions, whole sentences, and utterances in context.

2.3.2 Ontology

Ontology is a base of two Greek words “Ontos” which means “being” and “logos” which means “word”. Although generally associated with philosophy, the term is used in different fields. In Philosophy it is sub–category of “metaphysics” which means the study of the nature of reality. This simply means the study of what exists. Ontology in philosophy studies the nature of being and the categorization and interaction of the being. According to Barry Smith, “Ontology as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality.

The average man unconsciously is an ontologist in his approach. This is because the human mind organises and affects the human behaviour based on assumptions on the human minds view of the world. Take, for example, the knowledge that a visit to the hospital for the first time requires immediate registration, waiting in the waiting room before being attended to in the hospital. This is common knowledge, which unconsciously affects conduct in the hospital. This basic understanding of ontology transcends to other fields. For Artificial Intelligence (AI) systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them are reflected in the representational vocabulary with which a knowledge-based program represents knowledge.

In AI, ontology is generally defined as the explicit specification of a conceptualization (Gruber, 2009). Conceptualization is an AI term which refers to the “a set of objects which an observer thinks exist in the world of interest and relations between them (Genereseth and Nilsson, 1987). A more compositional definition given by Mizoguchi *et al. (*1997) describes ontology as a collection of concepts, the hierarchical arrangement of them and the interrelation between them.

In Computer Science, ontology is categorised into two, based on its role. It is categorised as a vocabulary and as a content theory. Ontology as a vocabulary, it is a representation vocabulary, often specialised to some domain or subject matter. Mizoguchi *et al. (*1997) defined ontology as a theory of concepts or vocabulary used as building blocks for

information processing. Ontology as a vocabulary the representation vocabulary provides a set of terms with which to describe the facts in some domain, while the body of knowledge using that vocabulary is a collection of facts about a domain (Chandrasekaran *et al*., 1999). This, in turn, does not mean the vocabulary is the ontology but in the conceptualization of terms, which the vocabulary is to be represented. Therefore language translation does not affect ontology but the concept in which the vocabulary is presented.

AI focuses mainly on content theories and mechanism theories. These two are equally important however without a good content theory to work with, a mechanism cannot work excellently. Ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge (Josephson *et al.*, 1999).

I. Types of Ontology

Guarino (1998), identified the following as types of ontology, Figure 2.2 presents a diagrammatical representation of types of ontologies.

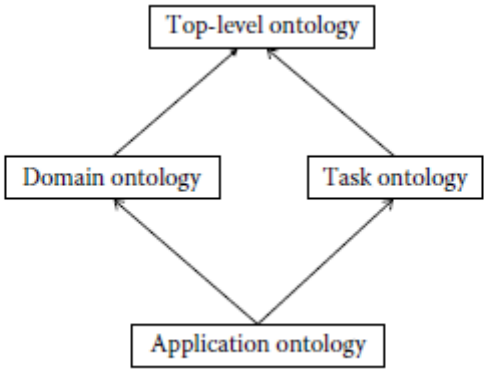


Figure 2.2: Types of Ontology
Source: Guarino (1998)

a) Top level Ontology

They are also known as Foundational or Upper-Level Ontology. They are used to describe general concepts and are applicable to various domains. They provide a basic description of objects, relations, events, and other elements of various domains. Upper ontologies try

to comprehensively capture knowledge about the world in general, describing for example space, time, object, event or action, and so forth, independently of a particular domain (Castaneda, Ballejos *et al*., 2010). Examples of top-level ontologies include Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) and the Basic Formal Ontology (BFO). A major benefit of upper ontology is the ability to support semantic interoperability among a large number of domain-specific ontologies by providing a common starting point for the formulation of definitions.

b) Domain Ontology

This refers to ontology for a specific domain or unique to an application. Domain Ontology defines how a group of users conceptualise and visualise some specific phenomenon. They aid in the development of object-oriented designs for applications and also building of building large knowledge systems.

c) Task Ontology

Task ontology is forms of problem-solving ontologies, which are aimed at describing or defining specific task or activity. Mizoguchi *et al. (*2010) classified the definition of Task ontologies into 2: (i) A subtask decomposition together with task categorization such as diagnosis, scheduling, design, etc. and (ii) An ontology for specifying problem-solving processes.

d) Application Ontology

Malone *et al*. (2010), defined application ontology as an ontology engineered for a specific use or application focus and whose scope is specified through testable use cases. They are descriptions of concepts that are often specialisations of Domain and Task Ontology.

II. Ontologies and Requirements Engineering

The success of a developed system lies in its ability to meet set goals and requirements. This can be dealt with at the requirements engineering stage of software development. At this level the goals, functionalities and requirements which are necessary to make the

software a success are identified. There are certain activities, which make up the requirement engineering process. Castaneda *et al.* (2010) summarise these activities as Requirements Elicitation, Representation, Analysis and Communication as shown in Figure 2.3. The use of ontologies in software engineering has gained popularity for two main reasons: (i) they facilitate the semantic interoperability, and (ii) they facilitate machine reasoning. There are certain characteristics that good requirements must possess. The absence of these characteristics produces difficulties in the management of the requirements process as a whole. Missing, incomplete or inconsistent requirements lead to faulty software designs, implementations and tests, which produce software of improper quality or safety risks. There are certain challenges encountered during the requirements engineering procedures, which create problems with the requirements engineering procedure. They include some of the following:

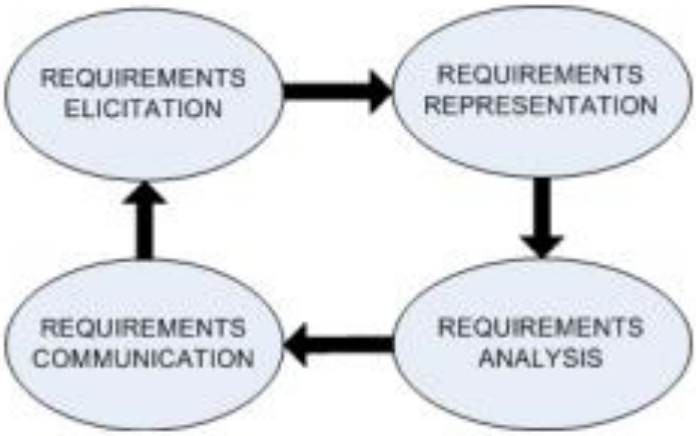


Figure 2.3: Requirements Engineering Activities
Source: (Castaneda *et al*., 2010)

a) Ambiguous Requirements

In the requirements engineering process, the customer's vague formulations need to be analysed and documented in a way that the software developers can implement the desired functionality of the product. User requirements are usually expressed in natural language (a major source of ambiguous requirements), which leaves a problem with the expressiveness, the completeness, and the accuracy of the statements. This occurs as a result of the interpretation by different stakeholders. Since many of these tasks are manual,

a skilled analyst is necessary. The analyst needs to write good specifications to avoid and eliminate these problems. (Korner and Brumm, 2009).

b) Inconsistency and Dynamic Requirements

The human factor (stakeholders) usually presents a constant change factor, which can be attributed to the desires and changing needs of the stakeholders. The inability of the requirements analyst to stay abreast and effectively manage these changes will lead to unsatisfied clients leading to an unsuccessful system (Castaneda *et al*., 2010). These changes also occur as a result of the multiple and conflicting requirements from different stakeholders. Requirements changes have a particularly significant impact on the consistency of specifications. Changes may introduce inconsistencies; and conversely, requirements changes may be necessary to handle existing inconsistencies. Therefore, the ability to handle inconsistent requirements is crucial to the successful development of requirements specifications (Castaneda *et al*., 2010; Nuseibeh and Russo 1999; Siegmund *et al*., 2011).

c) Incomplete Specifications

Incomplete specifications arise as a result different factors. Implicit requirements for example when not identified are left out. These requirements although not spoken are necessary and key to the success of the system as they equally affect the overall success of the system (Daramola *et al*., 2012; Onyeka, 2013). Non-identification can also result from a lack of experience of the Requirements Analyst. This contributes to developers' frustrations because they base their work on incorrect suppositions, hence the required product is not developed according to the desire of the user, which leads to dissatisfied customers (Castaneda *et al*., 2010).

In this era of the Semantic Web, the use of ontology is on the increase as a result of its ability to facilitate semantic interoperability and machine reasoning. Different researchers have suggested the synergy of ontology with the requirement engineering process. The ontology-driven requirement engineering approach has become a popular term. The

following are areas in which ontology have proven to be useful to requirements engineering (Dobson and Sawyer, 2006; Castaneda *et al.,* 2010):

i. It affects the requirements model itself by imposing and enabling a particular paradigmatic way of structuring requirements
ii. It enables the acquisition of structures for domain knowledge
iii. It is useful in the knowledge application domain
iv. It helps in the collection of Knowledge of the environment

Castaneda *et al. (*2010) gave a breakdown of some ontologies that has been applied to requirements engineering, these include Requirements Ontology (Castaneda *et al*., 2010), Requirements Specification Document Ontology (Hadad *et al*., 2009), Application Domain Ontology (Castaneda *et al*., 2010). The detailed descriptions are as follows:

a. **Requirements Ontology-** At the requirements elicitation/ specification stage, different forms of requirements are identified such as functional requirements, nonfunctional requirements, product requirements and others. The presence of this ontology aids in the reduction of ambiguous requirements and avoidance of incomplete requirements (Jayadianti *et al*., 2013; Castaneda *et al*., 2010) it serves as a tool for restrictions, verification and validation of requirements.
b. **Requirements Specification Document Ontology**-The use of ontology in the Software Requirements Specification (SRS) document aids in the reduction and elimination of insufficient specifications as well as the definition of the structure of the SRS document. It also serves as a source for knowledge reuse in situations of adaptations of SRS structure for requirements reuse for the similar and dissimilar applications. Object-oriented systems representing a useful analysis of a domain can often be reused for a different application program (Josephson *et al*., 1999).
c. **Application Domain Ontology**-The Application Domain Ontology (ADO) aids the development of the object-oriented design. ADO contains application domain knowledge and business information required for building software

applications in a specific domain. This ontology aids in the management of dynamism in the requirements (Castaneda *et al*., 2010).

III. Ontologies for Knowledge Management

Knowledge management is concerned with the representation, organisation, acquisition, creation, use and evolution of knowledge in its many forms. In this present knowledge era, given today's vast, complex and dynamic information environments, the potential for using information technology to help discover, deliver and manage knowledge is enormous (Jurisica *et al*., 2004). Knowledge management systems are common with large corporations as knowledge is considered the most important asset that enables sustainable competitive advantage in very dynamic and competitive markets. This makes knowledge management a crucial activity for many companies especially those based in knowledge-based economies. The major aim of Knowledge Management Systems (KMS) is to provide the right knowledge to the right people at the right time and in the right format, such that users can access and utilise the rich sources of data, information and knowledge stored in different forms (Lin and Wu, 2005).

Ontology is relevant to knowledge systems as it provides a shared and common understanding of a domain that can be communicated across people and application systems. For the effective management of knowledge, ontology plays an important role in enabling the processing and sharing of knowledge between experts and knowledge users (Sureephong *et al*., 2008). Going by its philosophical definition, ontology is the study of "what exists". For knowledge-based systems, what "exists" is exactly that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them are reflected in the representational vocabulary with which a knowledge-based program represents knowledge (Lin and Wu, 2005). Different researchers have proposed different forms of the ontology knowledge management system. (Jennings, 2000) proposed an agent-based knowledge based management system made up of three parts: the user interface, the search module and the knowledge generation module with the main objective of improving the capabilities of

industries to monitor, predict and respond to technological, product and market trends and changes. The system retrieved information from the web and other sources through the use of multi-agents and ontologies. The ontologies are used to specify the queries while the multi-agents search for information on the web and acting on a part of the original ontology. Brandt *et al.* (2008) proposed a system, which is a flexible ontology-based schema with formally defined semantics that enables the capture and reuse of design experience, supported by advanced computer science methods. Lin and Wu (2005), proposed a framework of ontology-based KMS that mainly focuses on performing the activity for projects and domain experts matching in which system architecture, ontology building, and semantic similarity calculation are addressed respectively.

IV. Ontology Learning

Ontologies are representations of reality, and as such, require frequent updates. This makes them expensive and difficult to maintain. Ontology learning was developed to solve this problem (Abel *et al*., 2015). The term "ontology learning" was coined by Madche and Staab (2002), it was based on the proposed set of tasks and methods for the automated generation of ontologies from natural language text including term extraction, taxonomy induction, and relation learning. In simple terms, ontology learning can be defined as the automatic or semi-automatic creation of ontologies, which includes the extraction of the corresponding domain's terms and the relationships between those concepts from a corpus of natural language text and encoding them with an ontology language for easy retrieval. Lehman and Volker (2014) stated that "supporting the construction of ontologies and populating them with instantiations of both concepts and relations, is commonly referred to as ontology learning." The primary focus of ontology learning is knowledge acquisition and this can be carried out using different approaches. According to Lehman and Volker (2014), the following are the various types of approaches to ontology learning:

a) Ontology learning from text

This refers to the automatic or semi-automatic generation of lightweight taxonomies by means of text mining and information extraction. This method involves the application of

natural language text analysis techniques to texts. This approach is based on the Ontology Learning Layer Cake (Cimiano, 2006). Many methods under this approach are inspired by previous work in the field of computational linguistics, essentially designed in order to facilitate the acquisition of lexical information from corpora. Although some ontology learning approaches do not derive schematic structures they focus on the data level. These methods derive facts from the text.

b) Linked data mining

This refers to the process of detecting meaningful patterns in Resource Description Framework (RDF). Being able to detect the structure within published RDF graphs can, on the one hand, simplify the later creation of schemata and, on the other hand, allow detecting of interesting associations between elements in the RDF graph (Lehmann and Volker 2014).

c) Ontology reuse

This refers to a process in which available (ontological) knowledge is used as input to generate new ontologies this involves the adaptation of existing ontologies to new domains by partially re-using existing schematic structures. This is based on the ideology that building an ontology from scratch is a resource-intensive process and the development of new ontologies does not tap the full potential of existing domain-relevant knowledge sources (Bontas *et al*., 2005; Lehmann and Volker, 2014).

V. Ontology Languages and Tools

a) Ontology Language

In simplistic terms, ontology language refers to formal languages used to construct ontologies. Maniraj and Sivakumar (2010), describes an ontology language as a formal language used to encode the ontology. It allows users to write explicit, formal conceptualizations of domain models. Antoniou and Van Harmelen (2004) stated the following as essential requirements of ontology language:

i. **Well-defined syntax**: this is a necessary condition for machine-processing of information

ii. **Well-defined semantics**: the Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. Therefore, ontology plays a key role in this effort, aiming at unifying, bridging and integrating multiple heterogeneous, international and multilingual digital content.

iii. **Efficient reasoning support**: the richer the language is, the more inefficient the reasoning support becomes. This leads to non-computability. Hence there is a need for a language supported by efficient reasoners and also expresses large classes of ontologies and knowledge.

iv. **Sufficient expressive power**: ontologies must be able to explicitly extend other ontologies in order to reuse concepts while adding new classes and properties. Ontology extension must be a transitive relation; if ontology A extends ontology B, and ontology B extends ontology C, then ontology A implicitly extends ontology C as well.

v. **The convenience of expression**: ontology should support the specification of multiple alternative user-displayable labels for the resources specified by an ontology.

Gomez-Perez *et al.* (2006) stated that the selection of a language for developing ontology is dependent on the preference of the developer as well as other surrounding factors, which include level expressiveness of a language, its underlying knowledge, representation paradigm and reasoning mechanism attached to it amongst others.

Ontology languages can be classified into 3 categories namely: Logical Languages, Frame-based Languages, and Graph-based Languages. Different ontology languages have been developed over the years. These include CycL, DOGMA, RDF, WebOnto, Graffo, OntoEdit, TODE, Hozo, Swoop, Top Braid, OWLGrEd amongst others (Barzdins *et al*., 2010; Falco *et al*., 2014; Mizoguhi *et al*., 1997; Sure *et al*., 2003). They differ based on structure, syntax as well as purpose.

b) Ontology tools

Over a number of years, environments and tools for building ontologies have grown exponentially. These tools are aimed at providing support for the ontology development process and for the subsequent ontology usage. Ontology tools can be applied to all stages of the ontology lifecycle including the creation, population, implementation, and maintenance of ontologies (Youn and McLeod, 2006). It requires users to be trained in knowledge representation and predicate logic. Ontology tools can be broadly divided into 3 namely: Web-based Tools, Computer-based Tools, and Client Server tools. An example of Web-based tools includes Ontolingua, OntoSaurus, WebODE, IKARUS, CO4, APECKS, SymOntoX. Computer-based tools include Protégé, KADS22, JOE, DOE, DUET, IODE, KAON Tool Suite, OCM, VOID, Apollo, OilEd, and Ontology Editor. Client-Server tools include OpenKnoME and LinKFactory. Ontolingua, for example, is based on the knowledge representation paradigm of frames and first order logic is the most complete of the ontology languages and the one considered as a *de facto* standard in the ontology community. Kalyanpur *et al.* (2006) developed SWOOP, which stands for Semantic Web Ontology Editor was originally developed as a tool for creating, editing and debugging OWL Ontologies. It presently has become an open source project with contributions from all over the world. It has reasoning support and provides multiple ontology environments in which ontologies can be compared, edited and merged. Another is Apollo, a user-friendly knowledge modelling application which is based on the basic primitives, such as classes, instances, functions amongst others. Apollo is not bound to any knowledge representation language and can be adapted to support different storage formats. This is via I/O plugins (Kapoor and Sharma, 2010). A more recent tool is PROPheT- PERICLES Ontology Population Tool. This is a novel application that enables instance extraction and ontology population from Linked Open Data (LOD) sources, such as DBpedia and Europeana, through a user-friendly graphical user interface. It is a novel instance extraction engine, for locating instances (realisations) of concepts and relations in a Linked Data source, filtering them and subsequently inserting them into a domain ontology. This tool allows the user to explore and retrieve existing knowledge in a repository of his/her choice. (Mitzias *et al*., 2016).

Another example Protégé 5.1.0, is a tool which allows a user to construct domain ontology, customise data entry forms and enter data. A recent tool is Graffo, which stands for Graphical Framework for OWL Ontologies. This is an open source tool that can be used to present the classes, properties and restrictions within OWL ontologies, or sub-sections of them, as clear and easy-to-understand diagrams. Another tool is Anzo. It is a tool which is used alongside Excel to generate an initial Ontology. It also includes a straightforward Ontology Editor, two rules Engines, workflow capabilities, Pubsub, and Integration with XML, DRFS, OWL, Relational Databases, and Web Services.

VI. Application of Ontologies in Requirements Reuse

There are different uses of ontology in requirements engineering, one of which is the representation of requirements. They include lexical-syntactic patterns in order to capture the semantics of the relationships in the domain. Lin *et al. (*1996), proposed a generic solution, which provides an unambiguous, precise, reusable and easy to extend terminology with dependencies and relationships among captured and stored requirements and also provides a great advantage for the reuse of requirements. Ontology-driven requirement engineering enables requirements specification that eliminates the presence of ambiguous requirements, which aids traceability of the requirements. Requirements traceability entails describing and following the life of software requirements in Software Engineering (Winkler and Pilgrim, 2010). In order to trace requirements to their sources and to the intermediary and final artefacts generated from them all over the development process, it is mandatory to consider and represent information related to their source and the requirements' history (Castaneda *et al*., 2014). This facilitates the use of requirements and other relational information. There have been different methodologies proposed on the use of ontology in requirements reuse.

Kossmann and Odeh (2010) proposed OntoREM, as an Ontology-driven requirements engineering methodology that was introduced in order to improve requirements quality while reducing the efforts (such as development and maintenance) for requirements reuse. Although the work of Kossmann and Odeh (2010) did not lay emphasis on requirement reuse, it is however important in the sense that it reports evaluation results of the proposed

method in a case study on aircraft operability domain, which increases the confidence in ontology-based methods in requirements reuse problem. Lopez *et al.* (2002) proposed a requirements metamodel to define reusable requirements. This metamodel enables requirement reuse in domains where semi-formal representations are used to capture requirements information. Gruninger and Lee (2002) proposed a requirements management process in the concept of reuse in product lines and shares the experience with the proposed method in embedded software industry. This study presented findings, which are efficient mechanisms for reusable requirements and reusable test cases development. Marrero *et al.* (2008) proposed a natural language retrieval system supported by a knowledge model as the reuse process implies a retrieval of stored requirements, but these requirements are generally expressed in natural language. Karatas (2012), proposed an ontology-based requirements reuse method for systematic reuse of product line requirements, which is composed of an ontology-based approach for reusable requirements definition and a requirements configuration approach for the reuse of requirements.

2.3.3 Analogy-Based Reasoning (ABR)

Analogy-Based Reasoning (also known as Analogical Reasoning) is the process of solving new problems based on the solutions of similar past problems (Lung *et al.*, 2007). This involves adapting from the source strategy and specifying according to the form of the target problem. There are different models and theories, which form the basis of this reasoning.

I. Models and Theories of Analogy

The term Analogy is derived from the Greek "Analogia" which means "proportion". There are two common objects in the analogy, the source and the target. It entails the transfer of knowledge/information from the source to the target. An analogy is a correlation between two entities or systems of entities that highlight aspects in which they are believed to be comparable. Analogical reasoning is based upon an analogy. A typical example is simply saying subject A is like subject B. An electrical battery is like a reservoir. "The reservoir" is serving as the base (the source) from which knowledge is generated and transferred and

applied to the target, in this context “An electrical battery”. These comparisons are based on a similarity between both subjects, which their ability to store power and transfer it when required.

Another way of defining analogy-based reasoning (ABR) is through the lens of problem-solving. It can be described as a process of solving new problems based on the solutions of similar past problems. A great way of understanding analogy is through mathematics. In trying to solve a problem in mathematics, past solved problems similar to the existing problem are viewed and studied to solve the new problem. This knowledge used in solving the past problem say A is used either the same way or adjusted to suit the uniqueness of this existing problem let’s say B. This is known as an analogous approach in solving problems. Analogy is used in various fields and one of its major benefits is that it aids in the development of a new procedure to solve problems as shown in the example above (Yu *et al*., 2014). However, in order to solve new problems, it is necessary that attention is placed on gathering information on past problems as without it; it is difficult to apply the knowledge adequately. This is the basic principle of the approach of ABR in the field of AI.

The form of an analogical argument is as given below (Copi and Cohen, 2005):

a. S is similar to T in certain (known) aspects.
b. S is said to have some other feature F.
c. Therefore, T can be said to also have the feature F, or certain feature F* similar to F.

Where S is the source domain and T is the target domain. A domain is formally made up of a set of objects, properties, relations and functions and an inferred set of statements about them. (a) and (b) are premises while (c) is called the inference of the argument. The form of the argument is inductive; the inference is not guaranteed to follow from the premises.

An analogy between S(Base Object1) and T(Target Object1) can be formally stated as a one-to-one mapping between objects, properties, relations and functions in S and those in T. Analogy could exist between some of the sets in S and T, not necessarily all the

corresponding sets. In practice, analogies are evaluated based on the most significant similarities (at times differences) that exist in both domains. Figure 2.4 shows a graphic representation of the structure mapping process of analogical reasoning.

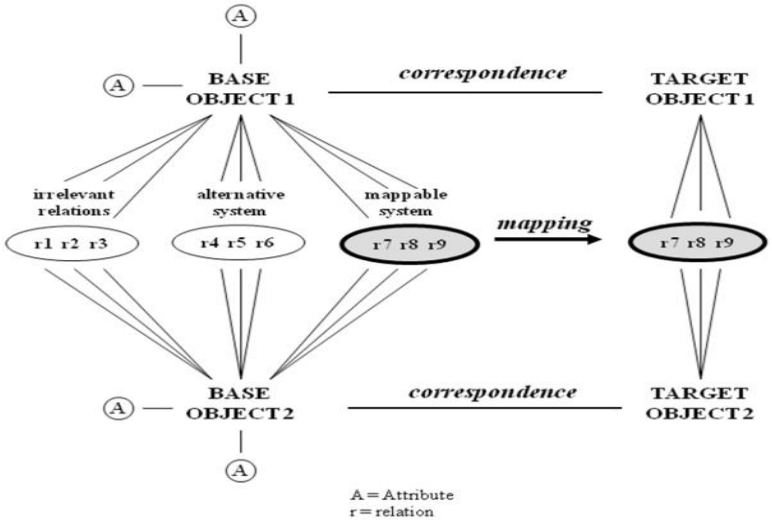


Figure 2.4: Diagram of the Structure Mapping Process of Analogical Reasoning
Source: (Stojkovic *et al.*, 2015)

Keynes (1921) further presented some terminology on analogy. They are as follows:

a) Positive analogy

Let K be a set of propositions $\{K_1, \ldots, K_n\}$ about a source domain S. Suppose that the corresponding set of propositions $\{K^*_1, \ldots, K^*_n\}$, abbreviated as K*, are all accepted as holding for the target domain T, so that K and K* represent accepted (or known) similarities. Then K is referred to as the positive analogy.

b) Negative analogy

Let N stand for a set of propositions $\{N_1, \ldots, N_j\}$ that is known to hold in S, and M* for a set $\{M_1^*, \ldots, M_k^*\}$ of propositions that are holding in T. Supposing the analogous propositions $N^* = \{N_1^*, \ldots, N_j^*\}$ fails to hold in T, and likewise the propositions $M = \{M_1, \ldots, M_k\}$ fails to hold in S, so that N, ~N* and ~M, M* represents accepted (or known) differences. Then N and M is referred to as the negative analogy.

c) Neutral analogy

The neutral analogy consists of a set of accepted propositions about S for which it is not known whether an analogue holds in T.

d) Hypothetical analogy

The hypothetical analogy is simply the proposition Z in the neutral analogy that is the focus of attention.

Different researchers have given theories and views on analogies dating back to the early days to the medieval days where the subject of analogy was greatly discussed (same meanings different words).

One of the foundational theories of analogy is the “Shared Abstraction” (Bao, 2008). This is as described by Plato and Aristotle. This is based on the view that analogies do not necessarily share any form of relationship but an idea or attribute, or philosophy but comparisons, metaphors and "images" (allegories) could be used as arguments, which are sometimes referred to as analogies. Hesse (1966) gave a refined version of Aristotle's theory, particularly centred on analogical arguments in the sciences. The author formulated three requirements that an analogical argument must satisfy in order to be acceptable i) requirement of material analogy i.e. observable similarities between domains; ii) causal condition i.e. a tendency to co-occurrence and iii) no-essential-difference condition. In suggesting an alternative approach of classifying analogical arguments, Bartha (2010) proposed the articulation model. The classification was on the basis of vertical relations within each domain as opposed to starting with horizontal relations. The central ideology was that a good analogical argument must satisfy two conditions: i) Prior Association which suggests that there must be a distinct link, in the source domain, between the known similarities (the positive analogy) and the further similarity that is expected to hold in the target domain (the hypothetical analogy); ii) Potential for Generalization which suggests that a reason must exist to think that a similar sort of link could be obtained in the target domain.

In Ancient Greek, analogy was understood as identity of relation between any two ordered pairs. This was supported by Kant's Critique of Judgment where Kant argued that there can be exactly the same relation between two completely different objects. A typical example of an Analogical question is this- HAND: PALM:: FOOT: ____; (sole). This relation is not based on the terminal definition palm and sole but based on the fact that they are both undersides of the hand and feet respectively, hence the similarity between the two (Gentner, 1983).

Modern research work expounded on this foundational framed view of analogy. A major theory developed is structure mapping theory by Gentner's (1983), which aims to capture the psychological processes that carry out analogical mapping.

Analogical mapping is the core process in analogy, which requires two aligning situations. A typical instance of analogical mapping includes a familiar situation which is the base or source description and is matched with a less familiar situation, which is the Target. The familiar situation suggests ways of viewing the newer situation as well as further inferences about it. According to Gentner (1983), the structure mapping theory refers to the comparison process (analogical mapping), which involves finding an alignment between the base and target representations that reveals common relational structure. Another theory is the Pragmatic Mapping theory given by Holyoak (1985). This theory views the analogical mapping processes as goal-oriented in nature, hence it views analogical mapping as a problem-solving process. This theory was further developed by Holyoak and Thaggard (1989) in the development of the Multi- Constraint theory. This theory defends structural mapping theory that the coherence of an analogy depends on structural consistency, semantic similarity and purpose.

II. Analogical Problem Solving

Analogical Reasoning is a source of knowledge and therefore it transcends to different domains from basic daily activities to Engineering. The transfer of knowledge between the source/ base and target shows the problems solving use of analogical reasoning. Analogical reasoning in problem-solving is dependent on the similarity of relational

structure between a known solved problem (source) and a novel problem (target). Different Researchers have shown the application of the problem-solving abilities of analogies (Duncker, 1945). “Radiation problem” is an example of Analogical problem-solving. It states - Suppose you are a doctor faced with a patient who has a malignant tumour in his stomach. It is impossible to operate on the patient, but unless the tumour is destroyed the patient will die. There is a kind of ray that can be used to destroy the tumour. If the rays reach the tumour all at once at a sufficiently high intensity, the tumour will be destroyed. Unfortunately, at this intensity, the healthy tissue that the rays pass through on the way to the tumour will also be destroyed. At lower intensities, the rays are harmless to healthy tissue, but they will not affect the tumour, either. What type of procedure might be used to destroy the tumour with the rays and at the same time avoid destroying the healthy tissue? This is an example of a problem (ill-defined), which creates an allowance for a creative solution. Duncker was able to propose solutions without using an analogy. Duncker proffered 3 possible solutions i) reducing the rays as they pass through a healthy tissue; ii) avoiding contact between rays and healthy tissues and iii) altering the relative sensitivity to rays of healthy tissue and tumour. Dunckers problem could be solved using analogies of past problem solved, which may be directly or indirectly related.

A major analogy used in tackling the above-stated problem is the attacker’s dispersion story. In this story, a General wishes to capture a fortress located in the centre of the country. With many roads leading up to the fortress, they have been mined such that only a small group of men can pass through each road and a large force will lead to a detonation. To tackle this problem, a divide and conquer strategy is applied. That is the army can split into small groups and attack from different routes. Although this story is parallel to the Radiation problem, an analogy is used from the Attackers dispersion (source base) to the radiation problem (target). Just like the divide and conquer strategy is used as an analogous solution, which simply involves multi-directing the rays from different points at the multi low intensity. This will leave the Healthy tissues unharmed and in the end destroy the existing tumour. Although totally unrelated, the necessary details can be applied to solving a problem. The human mind is replete of analogies hence its use in problem-solving.

Different Logicians and philosophers of science (Copi and Cohen, 2005; Moore and Parker, 1998; Woods *et al*., 2004) have given 'textbook-style' general guidelines that can be used for evaluating analogical arguments. A cross section of the important ones are as follows:

a. The more similarities (between two domains), the stronger the analogy.
b. The more differences (between two domains), the weaker the analogy.
c. The more the level of our ignorance about the two domains, the weaker the analogy.
d. The weaker the conclusion, the more acceptable the analogy.
e. Analogies involving causal relations are more plausible than those not involving causal relations.
f. Structural analogies are stronger than those based on superficial similarities.
g. The significance of the similarities and differences to the conclusion (i.e., to the hypothetical analogy) must be taken into account.
h. Multiple analogies supporting the same conclusion make the argument stronger.

III. Case-Based Reasoning

Case-Based Reasoning (CBR) is a problem-solving paradigm, which is able to utilise the specific knowledge of previously experienced, concrete problem situations (cases). This is applicable to different fields. For example, Lawyers make use of legal precedents in presenting their cases in court usually to seek justice in their favour. Another example is financial consultants, who while working on a difficult credit decision task, uses a reminding to a previous case, which involved a company in similar trouble as the current one, to recommend that the loan application should be refused. A case denotes a problem situation in CBR terminology. Solving a problem by CBR involves obtaining a problem description, measuring the similarity of the current problem to previous problems stored in a case base (or memory) with their known solutions, retrieving one or more similar cases, and attempting to reuse the solution of one of the retrieved cases, possibly after adapting it to account for differences in problem descriptions (De Mantaras *et al*., 2005).

Aamodt and Plaza (1994), provided what is regarded as a classic model of case-based reasoning as shown in the diagram. This represents the CBR cycle as shown in Figure 2.4. It includes the following steps:

i. RETRIEVE the most similar case or cases
ii. REUSE the information and knowledge in that case to solve the problem
iii. REVISE the proposed solution
iv. RETAIN the parts of this experience likely to be useful for future problem solving

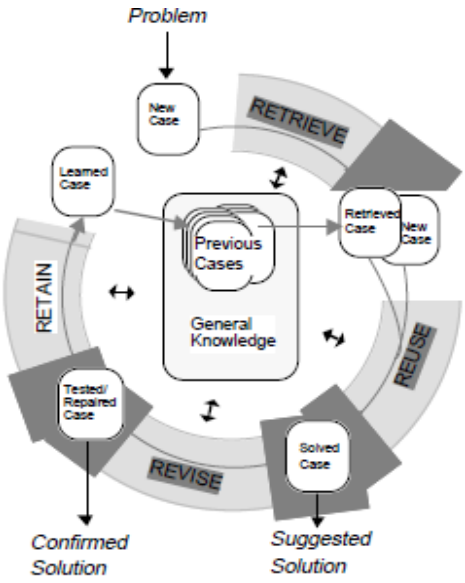


Figure 2.5: The CBR Cycle
Source: Aamodt and Plaza (1994)

IV. Analogy as a Paradigm for Specification Reuse

Different researchers have discovered reuse as a form of analogical problem-solving. This is because knowledge about existing systems embedded in specifications is transferred to specifications of new systems after identifying analogies between them. The reuse of specifications from the existing system in similar systems has its benefits. Amongst other benefits, it improves the completeness and correctness of specifications thus reducing the probability of improper design and implementation decisions with substantial delays and adverse cost effects in software projects (Spanoudakis, 1996). It exploits analogical similarities to identify ambiguity, incompleteness and inconsistency in new specifications as well as improve productivity during the costly requirements engineering phase of

software development. They can also provide a basis for communication by describing complex concepts in terms of well understood, existing specifications. Reusable specifications can provide solution templates for specifying new systems. This may be particularly beneficial for inexperienced software engineers because empirical studies of program design tasks suggest that novice software engineers do not have memory schemata to recall and are unable to structure and scope the domain space effectively (Maiden, 1991).

Requirements Specifications define the functions as well as necessary characteristics of a system. The presence of incomplete, inconsistent and ambiguous requirements presents the challenge of capturing the requirements and its elicitation. Without proper elicitation/specification of requirements, the functionality of the developed system is faulted. This can also create dissatisfaction with the user as well as incur high costs. It is, therefore, necessary that they are effectively managed and one of such ways is through requirements specification reuse. There are different forms of specification reuse. The specifications can be used in its original format for a similar system or can be adjusted to suit a less relative system (for example reversing the engineering code of an existing system for another system or larger applications). According to Maiden (1991), specification reuse across applications requires extensive customization to fit a reusable specification to the target domain.

2.4 SOURCES OF IMPLICITNESS IN REQUIREMENTS

This section gives a presentation of the various sources of implicitness in requirements as identified in literature and efforts that have been made to address them thus far.

2.4.1 Implicit Knowledge

There are implicit factors, which affect the performance of a system. Implicit knowledge (also known as tacit knowledge) is derived from Polanyi's Theory of Personal Knowledge, which states "*we know more than we can tell*". Another definition given by Nonaka *et al.* (2009) is that tacit knowledge refers to knowledge, which cannot be put into words. It simply refers to knowledge that is not known or communicated. According to Stone and Sawyer (2006), tacit knowledge exhibits the following characteristics:

i. Tacit-like knowledge may exhibit a presence in the requirements specification.
ii. The behaviour associated with tacit-like knowledge may already exist within the organisation in a physical or procedural way.
iii. The identification of tacit-like knowledge may impact on other requirements.

Tacit knowledge is crucial to the requirements elicitation stage, and Sawyer describes it as a problem and an advantage. It is a problem because if it stays implicit, it affects communication resulting into incomplete explicit knowledge. Tacit knowledge if not identified, will affect the performance of the user of the system. This is because it falls below the expectations of the user in terms of performance and other aspects of the system. Tacit knowledge can lead to challenges if not efficiently managed.

Tacit knowledge is the basis for tacit requirements. According to Jha (2009), tacit requirements or implicit requirements, are inexplicit requirements that are not directly expressed or captured but are essential to meet system's goal. In order to fulfil the objective of a system, it is essential that implicit requirements be made explicit as this can affect the overall performance of the system and the satisfaction of consumers of the system. According to Douglass (2009), IMR is included as a matter of professional duty. Identification of implicit requirements requires proper understanding and experience and touch on such subjects as implicit requirements, tacit knowledge, unknown assumptions and other implicit factors that are important to the functionality and acceptance of a system. However, as a result of their nature, they are difficult to manage and identify.

2.4.2 Outsourced Software

According to Deshpande and Richardson (2009), when a software organisation develops a product in a new domain or subcontracts the software to an external organisation with a different operational background via outsourcing this could bring about the emergence of implicit requirements. This in most cases is due to the fact that the external organisation to which the development of a software system or new domain is subcontracted is usually not present at the point of software elicitation or might realise other requirements essential to

the development of the software that was not explicitly stated thus bringing about implicitness in requirements engineering.

2.4.3 Ambiguity in Requirements Documents

In software development, ambiguity is considered a harmful aspect that needs to be eradicated (Garcia and Medinilla, 2007). Sinha and Husain (2016) describe ambiguity as for the biggest problem in the System Development Life Cycle of any software. According to Bussel (2009), ambiguity is best defined by 'having more than one meaning', and is inherent in natural language. Renkema (2004) attributed that ambiguity leads to noise in the communication channel. Based on these definitions, ambiguities or ambiguous requirements can be described as requirements that are unclear and vague, which pose a problem to the functionality of a developed system.

At the Requirements Specification Stage, the intended purpose of the software is defined. The requirements for the software are stated at this stage. According to Bussel (2009), a Requirements Specification (RS) is a written document in which an organisation writes out its understanding of a system prior to the actual design of the system. This stage provides clarity on the functions, requirements and expectations of the system to be developed. With Requirements Specification being a crucial basis for system development, it is essential that requirements stated at this stage be complete and concise. According to Berry and Kamsties (2004), Software requirements specifications need to be precise and accurate, to be self-consistent and to anticipate all possible contingencies. They also are not to be contradictory to one another. Errors at this stage will cause much more expensive errors in later phases of software engineering. A major source of errors is the ambiguity of the natural languages initially used to write the user requirements. Ambiguities are greatly associated with the use of natural language. As a result of the multiple meaning of words, it creates misconceptions, redundancy and errors, which will inherently affect the systems performance and consumer satisfaction. For this reason, they should be greatly avoided. Singh and Saikia (2015) classified ambiguities into four (4) types namely:

i. *Lexical Ambiguity*: ambiguity which occurs when a word has multiple meaning.

ii. *Syntactic Ambiguity*: this occurs when a given sequence of words can be given more than one grammatical structure and each having a different meaning. Example: SMALL CAR FACTORY. This sentence can have two meanings.— (small car) factory OR (small) car factory

iii. *Semantic Ambiguity*: It occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity. Example: ALL CITIZENS SHOULD HAVE A SOCIAL SECURITY NUMBER. The sentence can be interpreted as:

Every citizen has an individual social security number.

All citizens have same social security number.

iv. *Pragmatic ambiguity*: It occurs when a sentence has several meaning in the context in which it is uttered. It depends upon the background of the requirement engineers and thus has multiple interpretations. It influences the understanding of a phrase positively or negatively.

Example: *Do you want to have a cup a tea?* There are two meanings to this question. An informative question —

"*Do you feel a desire to a cup of coffee?*"

Or a polite offer

"*I can make you a cup of coffee if you want*".

Different researchers have proposed different methods of identifying or detecting ambiguities. Kiyavitskaya *et al*. (2008), proposed a three-step, semi-automatic method, which combines the strength of human reasoning and also automation methods that are supported by a prototype tool, for identifying inconsistencies and ambiguities in natural language requirements specification. Tjong *et al.* (2006), classified 3 approaches to detecting and resolving ambiguity in writing natural language requirements: i) approaches that define linguistic rules and analytical keywords; ii) approaches that define guideline rules, and iii) approaches that define language patterns. Gleich, Creighton, and Kof (2010) presented a tool that is able not only to detect ambiguities but also to provide explanations for detected ambiguities. Sinha and Husain (2016) proposed a concept of a tool, which aims at avoiding ambiguity at every stage of the SDLC of the software. The proposed tool

will be able to identify the ambiguous words and provide all the possible meanings of those ambiguous words clarifying the meaning of the sentence. Popescu *et al.* (2008) presented a three-step, semi-automatic method, supported by a prototype tool, for identifying inconsistencies and ambiguities in natural language software requirements specifications. The method combines the strengths of automation and human reasoning to overcome difficulties with reviews and inspections. First, the tool parses a natural language software requirements specification according to a constraining grammar. Second, from relationships exposed in the parsing, the tool creates the classes, methods, variables, and associations of an object-oriented analysis model of the specified system. Third, the model is diagrammed so that a human reviewer can use the model to detect ambiguities and inconsistencies. Schneider (2002) developed a new inspection technique, denoted Constructive Reading Inspection Process which is used to explore requirements inspection. It involves extracting the conceptual entities and their interrelationships as opposed to looking solely for defects.

Table 2.1 gives a summary of some tools that detects and resolves ambiguity based on the following parameters: approach used by the tool, technologies used, support for requirements per-processing, ambiguity concerned, the level of user interaction (viz. medium, low, high), and relevant information as remarks.

Some of these approaches used by this tools include:

I. Knowledge Based Approach

To extract and manipulate the meaning of the text, NLP system must have an extensive knowledge about the world and the domain of discourse. Knowledge-based approach applies this knowledge to resolve ambiguities. The knowledge-based system can be viewed as a search system that uses different types of knowledge with a view to constraining the search space. It offers optimal, acceptable and efficient search results. The Knowledge-based NLP systems are domain specific. To make it domain independent, language specific (i.e. lexical) knowledge and domain/world knowledge are separated, called ontology.

II. Controlled Language

A Controlled Language is used to reduce the ambiguities by restricted grammar and a fixed vocabulary. It improves readability and provides automatic semantic analysis. Languages are formed using grammar, e.g. grammar G = (V, T, S, PR), where V is a set of variables, T is a set of terminal symbols, S is a start symbol and PR is a set of production rules.

III. Style Guides

Style guides are used to avoid syntactic and semantic ambiguities by allowing inputting requirements in a specific manner (e.g. use active voice instead of passive voice, include braces, insert comma etc.). Sentence pattern is an alternative approach to avoid ambiguities where requirements are rewritten with minor modifications to match the predefined patterns.

IV. Heuristics Based Approach

This approach is based on machine learning, which makes use of corpus as an example for deducing further knowledge. Such approach usually guarantees a solution with high probability. In addition, employing heuristics, in these approaches, one can realise deterministic solution with a defined bound. The accuracy of the machine learning based system depends on the size of the corpus on which it is trained.

Table 2.1: Comparative Analysis of Ambiguity Resolving Tools

Feature Support	**Approach**	**Technologies Used**	**Pre-Processing**	**Concerned Ambiguity**	**User interaction**	**Remarks**
OOV of NLRS (Automatic) (Mala and Uma, 2006)	Knowledge based to ontology	Brill tagger, GATE tool	Yes	Pronoun Anaphora	Medium	Nonfunctional Requirements elicitation. Ontology generation.
RA in via OOM (Semi-automatic) (Popescu *et al*., 2008)	Controlled Language	Dowser parser	No	Semantic	Medium	Cannot deal with modal verbs and negations. Recall 78.8% (Compound-noun) Recall 93.9% (Single noun)
SREE (Semi-automatic) (Tjong, 2008)	Rule based, Style guide	WordNet, POS tagger	No	Identify Plural, Coordination, Pronoun, Quantifier, Vague	Low	Report summary of ambiguous and incomplete requirements statements. Recall 100% (w.r.t SREE's scope)
RESI (Semi-automatic) (Korner and Brumm, 2009)	Knowledge based to ontology	Stanford parser, Cyc, ConceptNet, WordNet	Yes	Avoid Lexical, Scope, Language Error	High	Input must be in the graph GrGen format.

NAI (Automatic) (Yang *et al*., 2010; Yang *et al*., 2011)	Machine learning/heuristics based	LogitBoost, Named entity recognition	Yes	Noun and Verb compound coordination, Anaphora ambiguity	Medium	Establish the degree of nocuity that the system should tolerate. Precision 70% and Recall 100% (Coordination) Precision 82.4% and Recall 74.2% (Anaphora)
SR-Elicitor (Automatic) (Umber *et al*., 2011)	Controlled Language	SBVR, POS tagger	No	Lexical, Syntactic, Scope-Quantifier	Low	SBVR rule generation. Recall 80.12% and Precision 85.76%
NL2OCL (Automatic) (Bajwa, *et al*., 2012)	Knowledge based to ontology	SBVR, Stanford parser	No	Attachment Homonymy	Low	A UML class model is required as an input. Recall 92.85% Precision 92.85% (Attachment) Accuracy 99.0% (Homonymy)
CKCO (Automatic) (Al-Harbi *et al*., 2012)	Knowledge based to ontology	WordNet, WSD	No	Lexical – Polysemy (ambiguity of nouns)	Low	Resolve ambiguity posed to a Question System. Precision 83.4%

Source: (Shah *et al*., 2015)

2.5 OVERVIEW OF RELATED WORK

This section reviews other research efforts that are related to the issues of implicit requirements in software engineering that have been reported in the literature.

Different researchers have developed various systems or tools, which are aimed at solving the problem of requirements management. In the past, prominent tools have been proposed for managing requirements in Requirements Engineering. However, these tools lack specific provisions for managing implicit requirements (IMR). An example of such tools includes CORE Enterprise, DOORS, Caliber-RM, RDD-100, RequisitePro, icCONCEPT-RTM, SLATE, Vital-Link and XTie-RT (Larsson *et al*., 2008; Choi, 2000).

Also, a number of other research efforts have been reported in the literature that has evaluated and reviewed implicit (tacit) knowledge and its effect on IMR. Some valuable works have developed techniques to expose sources of tacit knowledge during requirements elicitation and its negative effect on the quality of the requirements.

In Liddy (2001), a knowledge model that caters for capturing both implicit and explicit knowledge in the software engineering domain was proposed. The model integrates both explicit knowledge in the form of software artefacts and implicit knowledge in the form of arguments that constitute the context of the creation and validation criteria of captured knowledge. The work is limited in scope to a model that just captures both implicit and explicit knowledge. A method to highlight requirements that are potentially based on implicit or implicit like knowledge was proposed in Lang and Duggan (2001). The identification was made possible by examining the origin of each requirement, effectively showing the source material that contributes to it. It was demonstrated that a semantic-level comparison enabling technique was appropriate for this purpose. The work helped to identify the source of the explicit requirement based on tacit-like knowledge but it does not specifically categorise tacit requirements and its management. Mohammed (2008) proposed a method of tacit knowledge identification by solving pre-requirements specification tracing using statistical natural language processing techniques. A tool, which supports the identification of candidate cases of implicit knowledge, was developed. To

achieve this, an examination of the origins of requirements in the requirements specification was done by matching requirements to their respective sources in order to determine requirements that are not firmly derived from the source material, thereby reflecting an instance of either poorly sourced knowledge or tacit knowledge. However, the focus of the work is not to provide support for managing implicit requirement but identifying tacit knowledge in RE through pre-requirement tracing. Also, in MaTREx (Gacitua *et al*., 2009), a brief review and interpretation of the literature on the implicit knowledge that is useful for requirements engineering was presented. The authors described a number of techniques that offer analysts the means to reason about the effect of implicit knowledge and improve the quality of requirements and their management. The focus of the work was on evolving tools and techniques to improve the management of requirements information through automatic trace recovery; discovering the presence of tacit knowledge from the tracking of presuppositions and unprovenanced requirements; and the detection of nocuous ambiguity in requirements documents that imply the potential for misinterpretation. However, the focus of this thesis is more on managing IMR in RE while MaTREx deals more with handling implicit knowledge in RE. The relationship of the work to this is that implicit knowledge is just one of the sources of the emergence of IMR during RE.

An approach for modelling and managing tacit product line requirements knowledge was presented in Stone *et al.* (2006). The approach builds on modularizing variable feature requirements with aspects, using explicit join relationships for their integration semantics, modelling the commonality and the variability of the product line in a single aspectual model, describing details of the variability including variability constraints in tabular form, and visualising variability constraints graphically. It was further demonstrated that, with their approach, they could document various product line requirements knowledge explicitly that previously was distributed in the organisation and was handled implicitly, relying on the expertise and experience of the involved stakeholders. However, the model only caters for documentation of knowledge but does not provide room for the extracting IMR and its onward management.

A number of studies have aimed at a more practical approach of requirements reuse. These recent works include practical approaches to requirements reuse in product families. Stone *et al.* (2006), presented an incident report of requirement reuse in a case study of On-Board systems. The study aimed at discovering how requirements reuse can be integrated into DOORS. However, the focus was not to provide systematic support for Requirements Engineering within a framework.

In Singer *et al.* (2009) the application of rules derivation for the elicitation of implicitly expressed requirements in IT ecosystem was reported. By introducing rules into the infrastructure of the ecosystem, which is being observed for adherence by agents interacting in the system, deviations from these rules can be harnessed for finding potential candidates for new or changed requirements. These deviations are then processed using techniques like data mining and pattern recognition and then forwarded to requirements engineers for review. These implicitly expressed requirements are then leveraged to identify actual changes in the needs of the users of the systems, thereby enabling further advancements of the IT ecosystem. The emphasis of this work was the discovery of new IMR or changed requirements by using agents.

In Daramola *et al.* (2012), a system that uses semantic case-based reasoning for managing IMR was proposed. The model of a tool, which facilitates the management of IMR through the analogy-based requirements reuse of previously known IMR was presented. The system comprises two major components, semantic matching for requirements similarity and analogy-based reasoning for fine-grained cross domain reuse. This approach ensures the discovery, structured documentation, proper prioritisation, and evolution of IMR, which would improve the overall success of software development processes.

So far in the literature, there are not many empirical studies that focused specifically on the issue of implicit requirements within software organisations. The ongoing work reported in Dreyer *et al*. (2015) was done to identify the impact of tacit and explicit knowledge transferred during software development projects. An inductive, exploratory, qualitative methodology was applied in order to validate the tacit knowledge spectrum in software development projects. The work aims to create a conceptual model that supports future

software development projects in their tacit to explicit knowledge transfers. No concrete findings of the study were reported.

There are many studies that have addressed issues of requirements engineering within software organisations as a whole. For example in Quispe *et al*. (2010), the results of a diagnostic study of requirements engineering (RE) practices in very small software companies in Chile was presented. The study identified the state of the practice in these companies and the potential limitations that can hinder adoption of appropriate requirements engineering practices in the Chilean very small software enterprises. In Jantunen (2010), the report of an explorative study of software engineering practices in five small and medium-sized organisations was presented. Although the work did not focus particularly on RE practices, the study reveals interesting issues about software development practices in small organisations. In Aranda *et al*. (2007), a report of RE practices in seven very small scale enterprises (VSSE) in Canada was presented. The exploratory study found that RE practices in VSSE were diverse and are being successfully applied, the organisation's engaged experienced personnel in charge of their RE processes, requirements errors were rarely severe, and the organisations had strong cultural orientations. In Kauppinen *et al*. (2004), authors identified critical factors that affect organisation-wide implementation of RE processes. The work was based on a broad literature review and three longitudinal case studies that were carried out using action research. In Nikula *et al*. (2000), a study of the current RE practices, development needs and preferred ways of technology transfer of twelve small to medium-sized companies in Finland was reported. The study gave attention to the level of adoption for several RE practices and degree of adherence to general guidelines for RE practices.

Other surveys or field studies that focussed on requirements engineering practices in software organisations include Lubars *et al*. (1993) – requirements modelling; (Kamsties *et al*., 1998; Matulevičius, 2005; Solemon *et al*., 2010; Gorschek and Svahnberg, 2005) – adoption of standard RE practices; Rech *et al*. (2007) – intelligent assistance; and (Ihme *et al*., 2014; Thörn, 2010) – variability management. What is of note is that none of these previous empirical studies has focussed specifically on the management and handling of implicit requirements as discussed in this research.

2.6 THE CONTEXT OF THIS RESEARCH

From the foregoing issues, a number of gaps exist in the literature which defines the context of this research. The first is there is yet a lack of empirically proven evidence through research studies that have assessed the impact of IMR on the success or failure of software development projects. The second is the need for the generation of more elaborate approach that has the potential to address issues of managing IMR during RE. These two gaps become the premise for the central research question being investigated in this thesis, which is:

i. What is the impact of IMR on system development in practice?

ii. How can IMR be efficiently managed within an organisational context to promote successful RE during software development?

This thesis aims at proposing a viable solution to these questions.

2.7 SUMMARY

The chapter presents the issues that define the research context of this thesis. It started with a discussion on the important aspects of requirements engineering. It further looked at the core technologies that were used in this research, which are Ontology, Natural Language Processing and Analogy-Based Reasoning. Thereafter, the chapter reviewed sources of implicitness. The key aspects discussed include implicit knowledge, outsourced software and Ambiguity in the requirements document. The chapter closes by an overview of related work looking at the limitations of existing approaches and the gaps that this thesis attempts to fill.

CHAPTER THREE

METHODOLOGY

3.1 INTRODUCTION

This chapter describes the research methodology adopted by this thesis. The first part discusses the design and implementation of the empirical survey, while the second parts describe the architecture of the process framework for managing implicit requirements.

For the purpose of this study, the primary research methods used are empirical survey and design science research methods. This includes literature review, empirical survey, tool prototyping, case study and controlled experimentation.

For the empirical survey, a web-based questionnaire was drawn up. This questionnaire contained closed-ended questions and were distributed to fifty-six (56) participants from twenty-three (23) countries. The respondents are software developers of different companies, which fall into the category of small and medium-sized enterprises.

The survey investigated how IMR is managed within small and medium-sized software organisations and sought to understand the extent of consideration given to implicit requirements in the practice of software development by software developers and software development companies.

3.2 SURVEY RESEARCH DESIGN

In the survey, the applied research methodology used is quantitative research. This method is selected as a result of the nature of data gathered from the survey. The quantitative research is aimed at identifying the fundamental connection between empirical observation and mathematical expression of quantitative relationships (i.e. hypotheses). An overview of the adopted research process is presented in Figure 3.1.

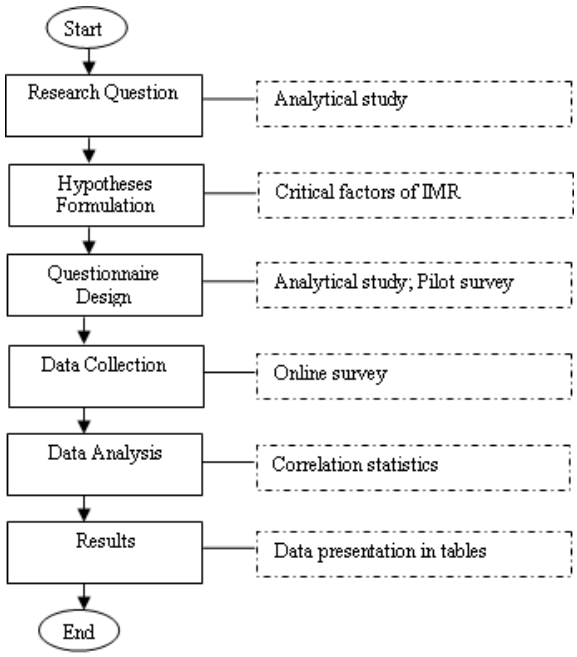


Figure 3.1: Overview of the Survey Research Process

The various components of this survey research process are as explained below.

3.2.1 Framework and Hypothesis Development

In this section, the proposed framework developed for the factors influencing IMR management during software development process in small and medium-sized enterprises is discussed. There are six factors that were considered in the framework as considered in Jantunen (2010), which include the number of years the organization has been in business, the size or number of the software development team, the scope of market operations of the organization; whether local, international or both, the professional status of the respondent, the personal experience of the respondent and that of the organization in RE. Figure 3.2 shows the proposed framework.

The Number of years in Business: this factor is selected to determine if the number of years of existence of a firm affects the knowledge and experience of the organisation in dealing with IMRs. The subject of IMR is one that is believed to be well-managed over time as experience is gathered over the years. This factor is to determine if its influence can be analytically proven.

Software Development Team Size: In a software development team, irrespective of its size, it consists of individuals with different levels of experience and knowledge level in the field of requirements engineering and implicit requirements. The selection of this factor is to assess whether truly there is power in numbers or it is more about the culmination of the knowledge of the implicit requirements irrespective of the size of the software development team.

The scope of the Market: the participation of organisations at different levels of markets determines their level of exposure to issues as they differ across the board. Although the Software Development Life Cycle (SDLC) of any software is fundamentally the same, the issues encountered might differ. Hence the need to determine if the scope of market that an organisation deals in, determines the kind of business

Professional Status: the position in the hierarchy of an organisation determines the kind of responsibility handled. This is also usually a determinant to the level of knowledge and experience in the field/ area of specialisation. This factor was selected to determine its influence on handling implicit requirements

Personal Experience in RE: Expertise in the said field is assumed to be directly proportionally to the knowledge in the field. However, this might not necessarily be true as expertise can be hindered by the level of exposure in the field and knowledge acquired. This factor is selected to determine if expertise in the field of RE has a direct influence on the knowledge of handling implicit requirements

Experience of the Organization: The level of experience of an organization in a field is believed to lead to acquired knowledge and expertise, which is shared with employees through possible training programmes, and other forms of interactive forums. This can have an influence in the organisations' policy in handling issues in the specialised field. This factor was considered to determine if the experience of the organisation influences the knowledge in the handling of implicit requirements.

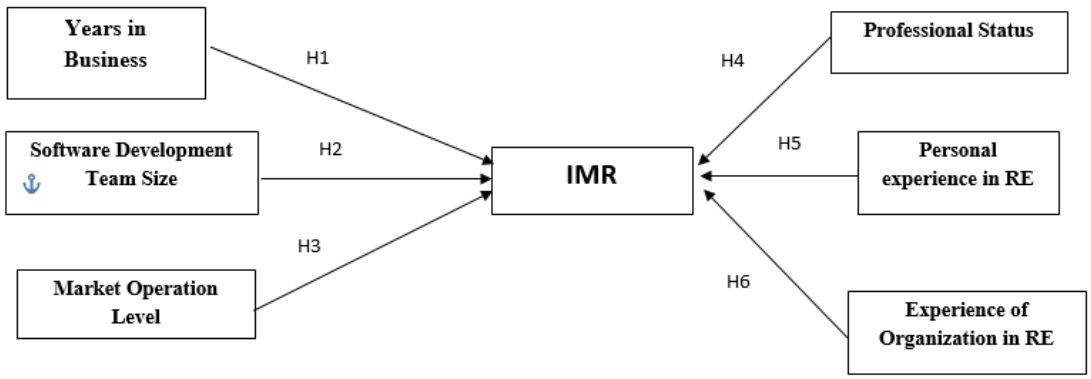


Figure 3.2: Framework for the hypothesis development

3.3 STRUCTURE OF THE SURVEY

The questionnaire was web-based. It had two pages containing two sections of questions to be answered by the respondents. The first section of the instrument contained introductory questions such as the name of the country where the software company is based, the name of the respondent's company or organisation and the professional background of the respondent. This section of the research instrument was used to gain information about the level of experience possessed by the respondent in the area of Requirements Engineering (RE) and the number of years that the respondents' company has had to engage actively in RE.

The second section of the questionnaire, however, contained close-ended questions that were aimed at eliciting information on the perception of implicit requirements within the respondent's organisation and how it is managed. The questions in this section sought to know the relevance attached to the deployment of implicit requirements in the software development process in respondents' organisations.

3.4 DATA COLLECTION METHOD

A web-based questionnaire was used to draw the participation of diverse respondents from different parts of the world. An open call was made through survey invites in relevant online requirements engineering and software engineering communities such as Yahoo Requirements Engineering Group, Linked-in Requirements Engineering Specialist group

(RESG), and Requirements Engineering Conference mailing list, AISWorld, and SEWORLD. This was to ensure that interested and qualified persons from these communities that have diversified global memberships were notified of the survey. Direct contact was also made with a few local companies in Nigeria, and some of the academic colleagues that are based in Europe and the US to help disseminate information about the survey. Many of them did this, by sending email invites to their colleagues within the software engineering community. The survey was online for a period of 6 months. At the end, 56 respondents participated in the survey, with respondents from 23 countries as shown in Table 3.1. The data collected from the online survey formed the basis of the analysis.

The questionnaire form is as shown in Appendix B, while the online survey questions and data are available at https://sites.google.com/a/covenantuniversity.edu.ng/resurvey1/. All of the respondents claimed to be software developers, with majority specialising in the development of business and enterprise software solutions.

Table 3.1: Number of Participants in each country

S/N	Country	No of Participants
1	Afghanistan	1
2	Australia	2
3	Austria	3
4	Brazil	2
5	Canada	3
6	Chile	1
7	Germany	4
8	India	5
9	Ireland	1
10	Israel	2
11	Italy	2
12	Macedonia	1
13	Netherland	3
14	Nigeria	2
15	New Zealand	2
16	Norway	2
17	Poland	1
18	Serbia	1
19	Spain	3
20	Sweden	1
21	United Kingdom	4
22	United States of America	9
23	Yugoslavia	1
	Total	**56**

3.5 TEST METHOD

The correlation analysis test, using the Spearman's rank correlation coefficient (Spearman's rho) was the major test carried out in this study. It was used to test each of the six hypotheses that were formulated and put forward in this study. For each hypothesis, the

correlation analysis technique was used to determine the relationship between certain factors and or characteristics of the respondents and their responses to the close-ended questions in the instrument. An investigation was also carried out to determine if the six factors or characteristics put forward in this study have any significant impact on the perception and handling of implicit requirements, and if so, the strength of the relationship.

The hypotheses are outlined below:

i. **H_1:** Number of years in business has a significant relationship with the knowledge and views of an organisation on implicit requirements.
ii. **H_2:** Size of software development team of an organisation has a significant impact on the knowledge and handling of implicit requirements.
iii. **H_3:** The organisation's scope of market operation has a significant impact on its knowledge and views on implicit requirements.
iv. **H_4:** Professional Status of an employee in an organisation has a significant impact on his/her knowledge and views of implicit requirements.
v. **H_5:** Years of personal experience of an individual in RE has a significant impact on the knowledge and views of implicit requirements.
vi. **H_6:** Experience of an organisation in RE has a significant impact on its knowledge and handling of implicit requirements.

3.6 SAMPLE PROCEDURE

Due to the nature of the study, purposive sampling is the type of sampling method employed in this study. The use of probability sampling, which is the alternative to the earlier method stated, was not employed in the study due to limitations arising from time and resources.

Contact was made with potential respondents via e-mails to seek their participation in the survey. An electronic mail was sent to each respondent explaining the purpose of the study. They were also be given a time frame of three (3) months within which the survey is to

take place which was later extended by another three (3) months to accommodate for more respondents.

Participants were also selected on the basis of their knowledge and practice of RE in their respective software development organisations.

A sample size of at least 50 participants was used for the survey for this study. The chosen sample size is to enable the effectiveness, efficiency and quality of the study.

3.7 DATA ANALYSIS AND RESULTS

There were 56 respondents (n=56) from different parts of the world. The data on the background of respondents as it pertains to the six factors is presented in section 3.2.1

Table 3.2 shows that a larger number of the respondents work for companies with over 20 years' experience (46.4%) in software development business, while 89.3% of the sampled population has more than 5 years' experience in software development. 19.6% of the respondents came from companies that have the international scope of operation; 42.9% from companies with a local scope of operation, while 37.5% described the operational scope of their company as both local and international. In terms of the professional status of respondents, 33.9% belong to the managerial level, 62.5% to a middle career level, while 3.6% belong to the lower level. The information in Table 3.2 shows that a greater percentage of respondents belong to middle-level personnel cadre compared to management and junior level employees.

Table 3.2: Data on Characteristics of Respondents

S/no	Factor	Analysis
1	Years of Business (years)	> 20 yrs = 26 (46.4%) 16 - 20 yrs = 6 (10.7%) 11 – 15 yrs = 9 (16.1%) 6-10 yrs = 9 (16.1%) 0-5 yrs = 6 (10.7%)
2	Software Development Team size (persons)	> 50 = 10 (17.9%) 21 - 50 = 5 (8.9%) 16 - 20 = 6 (10.7%) 11 – 15 = 9 (16.1%) 6-10 = 8 (14.3%) 0-5 = 18 (32.1%)
3	Scope of Market Operation	Local = 24 (42.9%) International = 11(19.6%) Both = 21 (37.5%)
4	Professional Status of Respondent's within their organization	Management level = 19 (33.9%) Middle level = 35(62.5 %) Lower level = 2(3.6 %)
5	Respondent's years of experience in RE	> 20 yrs = 15% 16 - 20 yrs = 2% 11 – 15 yrs = 21% 6-10 yrs = 34% 0-5 yrs = 28%
6	Experience of the Organization in RE	> 20 yrs = 18 (32.1%) 16 - 20 yrs = 5 (8.9%) 11 – 15 yrs = 9 (16.1%) 6-10 yrs = 14 (25%) 0-5 yrs = 10 (17.9%)

In terms of experience in requirements engineering (RE), 41% of respondents' organisations have at least 15 years of experience in RE, while 38% of respondents claimed to have more than 10 years' experience in RE practice. This showed that over 70% of the survey respondents used had been in RE Practice for an upward of 10years thereby giving the survey result good credibility.

3.7.1 Reliability Test

The reliability test was conducted in order to measure the consistency and stability of the data used for the analysis. The Cronbach's Alpha Test was used to determine the reliability of the data used in this study. According to Gliem and Gliem (2003), Cronbach's Alpha is a reliability test measure involving only one test administration to provide a given test with a unique evaluation. It is represented by the symbol α. During the process of establishing content validity of the questionnaire, a pilot survey was conducted using five (5) experts, who acted as a respondent in order to review the questions and offer suggestions for improvement. From the pilot survey conducted, it was observed that the initial questions were lacking in terms of the domain in which the organisations were developing their software products, the experience of the organisation in RE and finally if there exist specialised approach for handling IMR in their organisation. The revised questionnaire and additional suggested questions were then used in the survey instrument. The data collected is reliable under the Cronbach's Alpha test when α has a minimum of 0.7. For this study, the Cronbach's Alpha Test is valued at 0.783. This indicates that the data collected from the set questionnaire is suitable for carrying out further test and analysis.

Table 3.3: Reliability Test Table

Reliability Statistics	
Cronbach's Alpha	N of Items
.783	23

3.7.2 Hypothesis Testing

For the survey, Spearman Correlation Analysis was adopted to determine the impact of the six selected factors on the knowledge and perception of implicit requirements by software developers. The aim was to determine if certain factors have significant influence or relationship with the knowledge and perception of implicit requirements. The Spearman's Correlation Coefficient is a statistical measure of the strength of a monotonic relationship between two variables. It is represented by the spearman's rho (rs). In this study, the selected factors were tested against all the set of questions. However, the tables below

reflect responses which show the questions with significant relationship with the respective factor and all non-significant responses were excluded.

I. Number of Years in Business

H1: Number of years in business has significant relationship with the knowledge and view of implicit requirements

$H1_o$: Number of years in business has no significant impact on the knowledge and view of implicit requirements

The number of years in business represents the number of years in the practice of software development by a company. From the result extracted as shown in Table 3.4 the questions with the significant relationship are as listed below. Although there were a few significant relationships, they were however weak not exceeding 0.4. This means that there exists significant influence although it is not very strong.

Where:

Q2.7.1. A specialised approach, possibly with some automation support will be useful for managing implicit requirements (0.296)

Q2.14. Established RE management methods are adequate to handle implicit requirements for now (0.295)

Q2.6. Using experience plus tool support will be perfect for managing implicit requirements (0.379)

Q2.4. Implicit requirements does not have any impact on correctness of system architecture (0.295)

Q2.3. Implicit requirements does not have any effect on the acceptability of software product (0.344)

Table 3.4: Result of Correlation Testing for H1

			No. of years in Business
Spearman's rho	Q 2.14.	Correlation Coefficient	.295*
		Sig. (2-tailed)	.027
		N	56
	Q 2.7.1.	Correlation Coefficient	.296*
		Sig. (2-tailed)	**.027**
		N	56
	Q 2.6.	Correlation Coefficient	.379**
		Sig. (2-tailed)	.004
		N	56
	Q2.3.	Correlation Coefficient	.344**
		Sig. (2-tailed)	.009
		N	56
	Q 2.4.	Correlation Coefficient	.325*
		Sig. (2-tailed)	.014
		N	56
*. Correlation is significant at the 0.05 level (2-tailed). **. Correlation is significant at the 0.01 level (2-tailed).			

Table 3.4 shows that although the number of years in the business of software engineering has some effect on the knowledge and views of implicit requirements, there are other factors that affect the knowledge and perception of how implicit requirements should be handled in an organisation. The results of the analysis show that the greater the number of years in business the better the knowledge and perception of implicit requirements. This means that those with longer years in the business have a lot more regard for the subject of implicit requirements. It also shows that they recognise the need for improvement in the way implicit requirements are handled and its importance to the functionality of the developed system. Hence, H1 is accepted.

II. Size of Software development team

H2: Size of software development team has significant impact on the knowledge and handling of implicit requirements

$H2_o$: Size of software development team has no significant impact on the knowledge and handling of implicit requirements

The size of software development teams differs per company depending on the size of the organisation. In many instances, the larger the organisation, the larger the workload, and hence the need for a large software development team. The result of the analysis showed that the size of the software development team had a significant impact on the perception and knowledge of implicit requirements. However, these relationships are not very strong as none of the correlation coefficients exceeded 0.5 as shown in Table 3.5.

Table 3.5: Result of Correlation Testing for H2

			Size of software development
Spearman's rho	Q 2.8.	Correlation Coefficient	-.288*
		Sig. (2-tailed)	**.031**
		N	56
	Q 2.14.	Correlation Coefficient	.271*
		Sig. (2-tailed)	**.043**
		N	56
	Q 2.3.	Correlation Coefficient	.384**
		Sig. (2-tailed)	.003
		N	56
	Q 2.4.	Correlation Coefficient	.343**
		Sig. (2-tailed)	.010
		N	56
	Q 2.13.	Correlation Coefficient	.308*
		Sig. (2-tailed)	**.021**
		N	56
*. Correlation is significant at the 0.05 level (2-tailed). **. Correlation is significant at the 0.01 level (2-tailed)			

Where:

Q2.8. *Improper handling of implicit requirements can lead to poor system design and poor product performance (-0.288)*

Q2.14. *Established RE management methods are adequate to handle implicit requirements for now (0.271)*

Q2.3. *Implicit requirements do not have any effect on the acceptability of software product (0.384)*

Q2.4. *Implicit requirements do not have any impact on correctness of system architecture (0.343)*

Q2.13. *There is no need to evolve new methods to specially handle implicit requirements (0.308)*

The size of the software development team shows a positive correlation with questions Q2.14, Q2.3, Q2.4, Q2.13 with the exception of Q2.8, which had a negative value of (-0.288). This connotes that with an increase in the size of software development team the negative impact of implicit requirements on the correctness of system architecture, the acceptability of software product will reduce. Also, established RE methods will become adequate to handle implicit requirements, while reducing the size of software development team will increase improper handling of implicit requirements. From this analysis, it can be inferred that although the size of the software development team has a significant impact on the perception and handling of implicit requirements, there are other factors that also play a role since the values are closer to zero than to +1, which is a perfect positive correlation. Therefore, H2 is selected.

III. Level of Market Operation

H3: *The organisation's scope of market operation has significant impact on the knowledge and view of implicit requirements*

$H3_o$: *The organisation's scope of market operation has no significant impact on the knowledge and view of implicit requirements*

From the analysis conducted, the level of operation was classified based on the type of target market, which is local, global and both local and global. A larger percentage of the population of the respondents operate at either local level or at both local and global levels. The analysis showed that the target market of the company or level of operation of the organisation has no significant impact on the views and knowledge of implicit requirements. Hence, there is no table showing any significant relationship between any of the question, therefore H3 is rejected and $H3_o$ is selected.

IV. Professional Status in Organisation

H4: *Professional Status of an employee in an Organization has significant impact on the knowledge and view of implicit requirements*

$H4_o$: *Professional Status of an employee in an Organization has no significant impact on his/her knowledge and views of implicit requirements.*

The professional status of an employee within an organisation has been categorised into three levels. These are the Junior Level, Middle Level and Managerial Level. The analysis result in Table 3.6 showed that there was only one significant relationship between one of the questions and the professional status.

Table 3.6: Result of Correlation Testing for H3

			Q 8.	Q 2.5.
Spearman's rho	Q 8.	Correlation Coefficient	1.000	.347**
		Sig. (2-tailed)	.	.009
		N	56	56
	Q 2.5.	Correlation Coefficient	.347**	1.000
		Sig. (2-tailed)	.009	.
		N	56	56
**. Correlation is significant at the 0.01 level (2-tailed).				

Where:

Q8. *Your professional status in your organisation.*

Q2.5. *Relying principally on experience is sufficient for the discovery of implicit requirements during requirements elicitation (0.347).*

The result of the analysis showed that the higher the professional status, the greater the disagreement with the statement or close ended question. This means that those that are higher up in the career hierarchy do not believe that experience alone is sufficient for the discovery of implicit requirements. Although they agree that experience plays an important role, other approaches are required. Therefore, H4 is selected.

V. Years of Personal Experience in RE

H5: *Years of personal experience in RE has significant impact on the knowledge and view of implicit requirements*

$H5_o$: *Years of personal experience in RE has no significant impact on the knowledge and view of implicit requirements.*

The result of the analysis showed that years of personal experience in RE had a significant impact on some of the responses to the close-ended questions. These questions include the following:

Q8. *Your experience in Requirements Engineering (RE) practice in terms of years*

Q2.5. *Relying principally on experience is sufficient to the discovery of implicit requirements during requirements elicitation (0.290)*

Q2.6. *Using experience plus tool support will be perfect for managing implicit requirements (0.365)*

Q2.14. *Established RE management methods are adequate to handle implicit requirements for now (0.263)* Q 2.3 *Implicit requirements do not have any effect on the acceptability of software product (0.291)*

This analysis showed that although there is a significant relationship, it is however not strong as the coefficients are closer to 0 than +1, which is an indicator of a perfect positive correlation. The analysis in Table 3.7 shows that developers with longer years of experience have more regard and understanding of implicit requirements. This could be due to many practical cases of implicit requirements that they have handled during in the course of their career. Therefore, H5 is selected.

Table 3.7: Result of Correlation Testing for H5

			Q8
Spearman's rho	Q 2.5.	Correlation Coefficient	.290*
		Sig. (2-tailed)	.030
		N	56
	Q 2.6.	Correlation Coefficient	.365**
		Sig. (2-tailed)	.006
		N	56
	Q 2.3.	Correlation Coefficient	.291*
		Sig. (2-tailed)	.030
		N	56
	Q 2.14.	Correlation Coefficient	.263*
		Sig. (2-tailed)	.050
		N	56
*. Correlation is significant at the 0.05 level (2-tailed). **. Correlation is significant at the 0.01 level (2-tailed).			

VI. Experience of the Organization in RE

H6: *Experience of an Organization in RE has significant impact on the knowledge and view of implicit requirements*

$H6_o$: *Experience of an Organization in RE has no significant impact on the knowledge and view of implicit requirements*.

Q9. The experience of your organisation in RE.

The analysis showed that the level/years of experience of an Organization in RE have an impact on the knowledge and perception of implicit requirements. The results as shown in Table 3.8 shows that the years of experience of the Organization had a significant influence on 7 out of the 17 questions. They include the following:

Q2.5. *Relying principally on experience is sufficient to the discovery of implicit requirements during requirements elicitation (0.293)*

Q2.6. *Using experience plus tool support will be perfect for managing implicit requirements (0.373)*

Q2.14. *Established RE management methods are adequate to handle implicit requirements for now (0.397)*

Q2.3. Implicit requirements do not have any effect on the acceptability of software product (0.301)

Q2.4 *Implicit requirements do not have any impact on correctness of system architecture (0.314)*

Q2.15. *During requirements elicitation, stakeholders deliberately withhold certain information, which creates implicit requirements scenarios (0.387)*

Q2.13. *There is no need to evolve new methods to specially handle implicit requirements (0.297)*

Table 3.8: Result of Correlation Testing for H6

			Q 9
Spearman's rho	Q 2.13.	Correlation Coefficient	.297*
		Sig. (2-tailed)	.026
		N	56
	Q 2.14.	Correlation Coefficient	.397**
		Sig. (2-tailed)	.002
		N	56
	Q 2.15.	Correlation Coefficient	.387**
		Sig. (2-tailed)	.003
		N	56
	Q 2.3.	Correlation Coefficient	.301*
		Sig. (2-tailed)	.024
		N	56
	Q 2.4	Correlation Coefficient	.314*
		Sig. (2-tailed)	.018
		N	56
	Q 2.5.	Correlation Coefficient	.293*
		Sig. (2-tailed)	.028
		N	56
	Q. 2.6.	Correlation Coefficient	.373**
		Sig. (2-tailed)	.005
		N	56
*. Correlation is significant at the 0.05 level (2-tailed).			
**. Correlation is significant at the 0.01 level (2-tailed).			

The results of the analysis show that companies with longer years of experience in RE acknowledge the importance of implicit requirements, regards them as crucial to the functionality of a system and that they have an effect on the consumer satisfaction. Although there is a significant relationship, the relationship is however not a strong one as it is below 0.5. With the correlation coefficients closer to zero, this indicates a weak relationship. This implies that there are other factors, which play a major role in the knowledge, understanding and view of implicit requirements. Hence, H6 is selected.

3.8 DISCUSSION

Based on the outcome of the analysis of the result of the survey, four salient issues can be identified, which shall now be discussed. First, it was observed that there are critical organisational factors such as number of years in business of an organisation, and the years of experience of an organisation in dealing with RE, and size of software development team that have a positive correlation with the views, and handling of implicit requirements within an organisation. From this, one can safely argue that the level of maturity of the software process in an organisation will affect the way implicit requirements are managed, although high maturity of software process may not automatically translate to handling implicit requirements the right way because of the existence of other factors. Also, the scope of operation of an organisation whether local or global is a key determinant factor of how well an organisation handles implicit requirements. Second, there are critical human factors such as the general professional experience of employees, and the level of experience in RE that determines the way implicit requirements are perceived and managed within an organisation. Therefore, it is safe to say that organisations that have persons with significant professional experience in software development and RE in managerial positions, and also a significant bunch of these type of personnel in mid-level positions are more likely to perform better in terms of handling of implicit requirements than those where this is not the case.

The result of this survey also points to the fact that although so far the use of experience has played a significant role in handling implicit requirements, a significant number of practitioners believe that additional means that can complement the use of experience such as tool support are necessary. There also exists a significant number of practitioners that believe that existing requirements management tools are sufficient to handle implicit requirements for now, if maximised, and there is no need for new tools. In addition, there is a consensus that implicit requirements are real, and there are many deliberate situations caused by users that lead to the emergence of implicit requirements exist.

The findings from this survey revealed a number of issues and claims by respondents that need empirical verification by the requirements engineering community. For example, it

will be interesting to ascertain the strength of specific RE tools to manage implicit requirements in terms of addressing specific concerns across the RE lifecycle such as the discovery of hidden requirements, analysing implicitness, traceability, prioritisation and change impact analysis of implicit requirements. Also, a comparative evaluation of the existing support tools for implicit requirements is necessary in order to validate the potential of these tools to solve existing challenges and ascertain gaps that still exist.

3.9 DISCUSSION OF VALIDITY THREATS

The results obtained in this empirical study needs to be understood within the strengths and limitations of the selected research methodology. Hence, in this section how this study addressed specific validity threats are explained.

Conclusion Validity: this refers to whether the right conclusions can be drawn about the relationship treatment and the result obtained from the survey. Some of the concerns addressed in this aspect of validity are:

Low statistical power: In a highly technical domain such as requirements management, having a large number of respondents is not so much of a strength as identifying persons that are truly knowledgeable on the issue of managing implicit requirements. The 56 respondents that are located in 38 distinct organisations and across 23 countries is sufficient for a small scale empirical studies that seek to give a first empirically based opinion on the handling of implicit requirements in the industry.

Reliability of measure: the spearman‘s correlation coefficient that was used to investigate the relationship between the variables in the stated hypotheses (H1-H6) is a standard statistical measure that is suitable for the task it was used to perform. Also, in order to enhance the reliability of the measuring instrument, a pilot study was conducted initially, which improved the quality of questions.

Reliability of treatment: all respondents had the same kind information. The questions were in English, which happens to be the main language for business in the respondents’ organisations.

Construct validity: this refers to the extent to which the operational measures that are studied truly represent the theoretical constructs on which those operational measures were based (Wohlin *et al*., 2012). To achieve this, all respondents had the same instructions as a guide for completing the questionnaire. The task was the same for all, which is to complete the online questionnaire. Hence, the results obtained from the survey depends on only one variable, which eliminates any mono-method bias effect.

Internal Validity: this refers to whether other factors other than the treatment influenced the outcome of the survey. For the survey, all respondents were software practitioners who claimed to have ample experience in requirements engineering. The bulk of participants were recruited from professional online communities such as Linkedin Requirement Engineering Specialist Group (RESG), Yahoo Requirements Engineering Group, SEWORLD and AISWORLD. Generally, the respondents have significant experience in RE with 38% having more than 10 years' experience, while 72% have more than 5 year experience in RE. Also, they were given sufficient background introduction, which they had to read before the questions were presented to them.

External Validity: The key interest of this aspect of validity is whether we can generalise the outcome of the survey to a larger context. The respondents have mostly experienced software engineers, who have practical experience on issues that deal with implicit requirements and located in different parts of the world. A concern could be that possibly the result would have been different results if a larger pool of qualified respondents was used for the survey. However, we waited six months to have the 56, it could not be ascertained if the number would have been significantly more if we have waited for a longer time. Although, we do not consider this as a major threat to the reliability of the outcome of this survey, an interesting point for future study is to have a wider group of requirements engineers participate in the survey.

3.10 REQUIREMENTS OF THE PROCESS FRAMEWORK

From previous work done by authors (Fabbrini, *et al.,* 2001; Kamsties *et al.,* 2001; Lami *et al.,* 2004; Meyer, B. 1985; Wilson *et al.,* 1997), the following are instances or scenarios that could possibly make a requirement implicit:

i. The occurrence of ambiguity in a requirement statement;
ii. The presence of vague words and phrases;
iii. The presence of vague imprecise verbs;
iv. The presence of weak phrases; and
v. The occurrence of incomplete knowledge in a requirement statement.

Hence, these factors enable the fulfilment of the second objective of the proposed framework for managing IMR.

3.11 PROCESS FRAMEWORK FOR MANAGING IMR PROCESS

Based on the identified requirements this thesis proposed a framework that is shown in Figure 3.3. The components of the framework are presented in the sections following.

3.11.1 Components of the Framework

The framework proposed in this thesis integrates three core technologies NLP, ABR and ontology for discovery and managing of IMR. The architectural view of the process framework is presented in Figure. 3.3. The core system functionalities are depicted as rectangular boxes, while the logic, data and knowledge artefacts that enable core system functionalities are represented using oval boxes. A detailed description of the framework is given below.

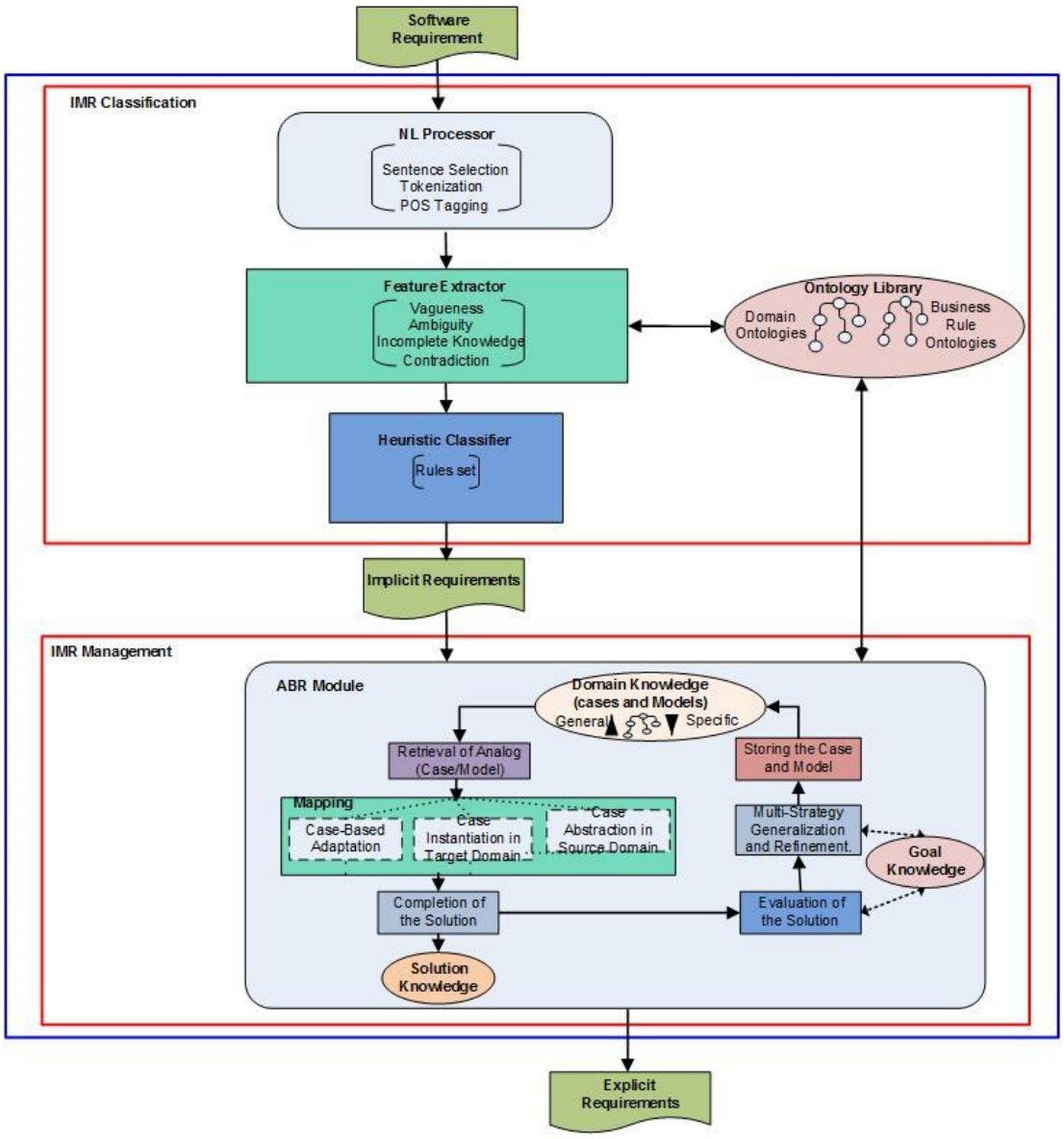


Figure 3.3: IMR Process Framework

I. IMR Identification and Extraction

In this section, the part of the framework that deals with knowledge representation and extraction are described.

a) Data Preprocessing

A preprocessed requirements document is the input to the framework. Preprocessing is a manual procedure that ensures that the requirements document is in the required format

acceptable for use in the system. This entails extraction of boundary sentences from the requirements document and further representing images, figures, tables etc. in their equivalent textual format.

b) NL Processor

The NL processor component facilitates the processing of natural language requirements for the process that enables feature extractor. The core natural language processing operations implemented in the architecture are as follows:

a) *Sentence selection*: This helps in splitting the requirements statements into sentences for onward processing.
b) *Tokenization*: This further splits the requirements sentences into tokens. Tokens are usually words, punctuation, numbers, etc.
c) *Parts of speech (POS) tagging*: This classifies the tokens (words) into parts of speech such as noun, verb, adjective and pronoun.
d) *Entity Detection*: The process of dividing a text into syntactically correlated parts of words, like noun groups, verb groups, but does not specify their internal structure, nor their role in the main sentence.
e) *Parsing*: This creates the syntax tree which represents the grammatical structure of requirements statements, in order to determine phrases, subjects, objects and predicates.

The Apache OpenNLP library[1] for natural language processing was used to implement all NLP operations.

c) Ontology Library

The ontology library and ABR modules, make up the knowledge model of the process framework. The ontology library serves as a storehouse for the various domain ontologies (.owl/.rdf). The domain ontologies are those that have been developed for a specific

[1] https://opennlp.apache.org/

purpose or those of business rules. The ontology library was implemented using Java Protégé 4.1 ontology API, while the ABR module was implemented using the concept proposed by Maiden (1991).

d) Feature Extractor

The feature extractor heuristic gives underlying assumptions for identifying potential sources of IMR in a requirement document. Due to semantic features on which natural language text exist and by taking into account previous work done by authors such as (Fabbrini *et al*., 2001; Kamsties *et al*., 2001; Lami *et al*., 2004; Meyer, B. 1985; Wilson *et al*., 1997), the following characteristic features underline what to look out for in a text in terms of surface understanding that could possibly make a requirement implicit:

a) Ambiguity such as structural and lexical ambiguity.
b) The presence of vague words and phrases such as “to a great extent”.
c) Vague imprecise verbs such as “supported”, “handled”, “processed”, or “rejected”
d) The presence of weak phrases such as “normally”, “generally”.
e) Incomplete knowledge.

Based on this underlying characteristics, the feature extractor is made up of various algorithms (lexical, pragmatic, syntactic, vagueness, incomplete knowledge) and also repository (weak phrases, vague words and phrases) of keywords that identify potential IMR.

II. Implicit Requirements Management

This section describes the parts of the architecture that deals with IMR management and its knowledge reuse.

a) ABR Module

The ABR component facilitates the knowledge reuse capability of the framework. This component is influenced by Maiden (1991), as stated earlier which consist of three type of

knowledge (domain, solution and goal). These three dimensions have been considered in the formulation of the Implicit Requirements Model (IRMM).

In order to manage IMR, a reuse-based IRMM is outlined below, which is a formal representation of requirements that create a basis for the reuse of implicit requirements associated with existing requirements in order to discover the implicitness of new requirements. This formal representation is an extension of the formalisation presented in Daramola *et al*., (2012).

The IRMM is an eight-tuple denoted as IRMM = < D, S, G, O, R_{id}, RQ_i, IMR_{id}, IMR_i > where

D is a description of the domain of the software project;

S is a description of the solution approach adopted by software project;

G is the goal of the system under development;

O is a description of the domain Ontology of the Requirement R;

R_{id} is the unique id of the requirement; and

RQ_i is the requirement statement represented by R_{id};

IMR_{id} is the unique id of the implicit requirements associated with R_{id};

IMR_i is the description of implicit aspects associated with the requirement RQ_i denoted as R_{id}.

The goal of the IRMM is to provide a uniform structure for describing requirements such that it will be possible to establish a basis for analogy reasoning. A case-based representation of requirements will classify the known parts of IRMM as problem specification of a case at hand, while the unknown part will constitute the solution part. From our IRMM, the set {D, S, G} represent the domain, solution and goal parts of both the source and target project.

b) ABR Model and Similarity/Difference Measure

The ABR model is partitioned into a problem part and a solution part. Both parts of the model shares a common domain abstraction, which is made up of the eight-tuple of the IRMM model. In order for analogical matching to be performed, the source domain, target domain and their domain abstractions must share a coherent structure of semantically equivalent facts. The extent of the analogical match is determined by the degree of overlap between these mapped structured facts. At the instance of a new (target) case, the *analogical matcher* identifies candidate analogical matches with one or many domain abstractions using a semantic matching algorithm (S-matcher) to compute the similarity between the problem parts of the new case and all existing relevant cases in the case library to determine suitable candidates for retrieval. The matching task is either at the element level or structure level (Yatskevich, and Giunchiglia, 2004).

Element level matchers consider only the information at the atomic level of both and returns the semantic relation that exist between them (e.g. equivalence, disjoint, part-of, kind-of) while structure level matchers take into consideration also the information about the structural properties of both schemas to determine the whole similarity coefficient, which is usually between ([0-1]). In carrying out the similarity matching for requirements similarity, a general knowledge base or upper-level ontologies, WordNet was used as the concept hierarchy. The model also supports using an existing domain ontology as concept hierarchy where such an ontology already exists or can be developed. The selected concept hierarchy then provides the basis for computing the semantic relatedness of two requirements. The solution part of a chosen retrieved case is then used verbatim or revised as the solution part of the target case. Figure 3.4 shows a flowchart of the process for analogical matching used in the ABR model. Firstly, the analogical matcher identifies candidate analogical matches with one or many domain abstractions. The abstraction selector then reasons heuristically about key differences between these abstractions to select the best match. Thirdly, the analogy determiner combines quantitative measures of similarity from the analogy matcher and selector to determine the degree of overall analogical match.

An example of a network demonstrating the structural isomorphism of the analogical match between a University Course Management System and a Smart City Parking System is shown in Figure 3.5. These two domains are case projects used in this study (oval shapes represent domain objects, rectangles and lines show domain terms). The possible reuse that can be exploited from this analogy is the structural and functional details such as processes (e.g., "sensor car park" and "course placement"), data stores (e.g., "sensored parking space" and "course place") and external agents ("driver" and "student"). Although both systems are in different domains, they both share significant surface features (e.g., reservations, waiting lists, places) which assist analogical recognition and understanding.

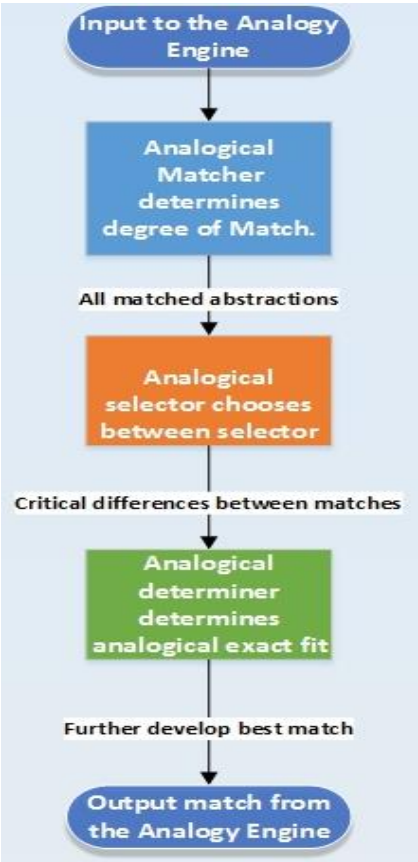


Figure 3.4: Flowchart for Analogical Matching

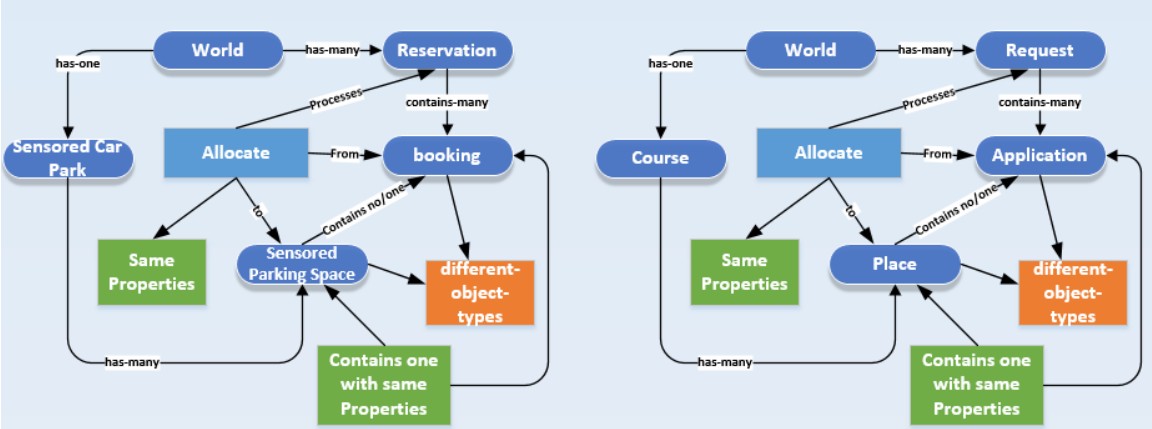


Figure 3.5: An Example of a Structural Isomorphism Network between Two Domains

3.12 SUMMARY

This chapter presents the research methodology adopted by this thesis. The first part has discussed the design and implementation of the empirical survey, while the second part has described the architecture of the process framework for managing implicit requirements.

From the first part, the result of the empirical survey, the result of this survey points to the fact that although so far the use of experience has played significant role in handling implicit requirements, a significant number of practitioners believe that additional means that can complement the use of experience such as tool support are necessary. Also, the fact that no other empirical study so far has looked specifically at the issue of implicit requirements makes the outcome of the survey potentially valuable to practitioners.

From the second part, an architectural framework that integrates three core technologies NLP, ABR and ontology for discovery and managing of IMR has been proposed for onward implementation and evaluation.

CHAPTER FOUR

SYSTEM DESIGN AND IMPLEMENTATION

4.1 INTRODUCTION

This chapter presents a description of the design and implementation of the support tool-PROMIRAR (PROduct for Managing Implicit Requirement using Analogy-based Reasoning) for identification and managing IMR.

4.2 MOTIVATION FOR PROMIRAR

The vision of PROMIRAR originated from the way implicit requirements are handled by requirements engineers who use their initiative and experience to address the challenges that the absence of such requirements pose to the overall purpose and functions of the system.

PROMIAR is essentially a prototype implementation of the architectural framework as given in Chapter 3.

4.2 Performing IMR Management with PROMIRAR

The process of using the PROMIRAR is as follows:

Step 1: The requirements document is preprocessed to get the requirements in a text file format, without tables, images and graphs.

Step 2: The available domain ontologies are selected or a new one is created semi-automatically for further use in the IMR identification.

Step 3: The requirements documents and the domain ontologies are imported into the PROMIRAR environment.

Step 4: The analyse button is clicked to permit the feature extractor to recognize likely sources of IMR in the requirement document.

Step 5: The prospective IMR are flagged by PROMIRAR using the heuristic classifier module.

Step 6: The analogy engine is called. If existing analogy exists, the best match for the supposed IMR to be explicated is selected and returned to the user for possible modification or direct use.

Step 7: Expert approves/disapproves the recommendations by either accepting/rejecting the recommendations by PROMIRAR.

Step 8: Each approved IMR and its explicated part are then added to the case base of PROMIRAR. Requirements that do not have recommendations for explicating the requirement, the ABR Module is called.

A flowchart of the above steps is a shown in Figure 4.5.

4.3 SYSTEM MODELLING FOR PROMIRAR

In this section, a full description of the system design using different models of the Unified Modelling Language (UML) was given below which includes:

1. **Use Case**: This is a demonstration of the interaction a user has with the system. It illustrates the association between the different use cases and the user's involvement. The use case diagram identifies the different use cases and the various types of users of the system.
2. **Class Diagrams**: This is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
3. **Activity Diagram**: This is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Activity diagrams show the overall flow of control.
4. **Sequence Diagram**: This is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. It shows object interactions arranged in time sequence.

4.3.1 Use Cases for PROMIRAR

Use cases specify the functionality that the system will offer from the user's perspective. A use case specifies a set of interactions between a user and the system to achieve a particular goal. Unlike cases where the principal actors are more than one, which may include the user and the administrator. PROMIRAR was designed as a plug-in tool which has only one actor which is the user of the application. The use case diagrams summarise graphically, the interactions of the user with the PROMIRAR.

The use case diagram in Figure 4.1 models the functionality that the PROMIRAR tool provides from the perspective of the user of the system who is the requirements engineer. The following are identified functions from the system use case model.

i. Open SRS file
    a. Search File Browser for SRS Document (text file)
ii. Edit Text file
    a. Edit Text
    b. Save Text
iii. Select Output File Directory
    a. Search File Browser for Output Directory
iv. Ontology Management
    a. Create new Ontology
    b. Import existing Ontology
v. Select Analysis
    a. Lexical (General)
    b. Lexical (Context Disambiguation)
    c. Vagueness
    d. Incomplete Knowledge
vi. Start Analysis
vii. Logs
    a. View Log
viii. Help

a. Check For Updates
b. About PROMIRAR
c. About Analysis Types

The system use case for the PROMIRAR can be shown as below according to the functions available to the actor (The User):

SYSTEM USE CASE

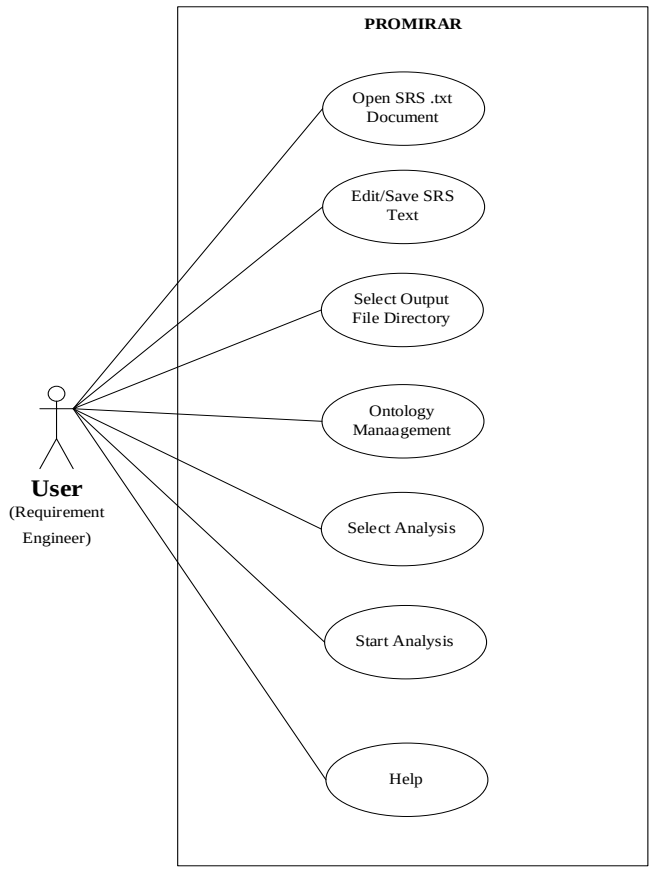


Figure 4.1: Combined Use Case of PROMIRAR

The Use Case Narratives of specific use cases of PROMIRAR are presented in Tables 4.1-4.8.

Use Case Narrative 1(Import SRS): this use case will enable PROMIRAR to import SRS document in text format.

Table 4.1: Use Case Narrative for Importing SRS Document

Use Case 1	**Open SRS .txt Document**		
Goal in content	User would be required to Open an SRS .txt file to be analysed		
Level	This is a basic Open File use case		
Parameters	In: SRS .txt File Out: Read and Output file Content		
Preconditions	The file must be a .txt file. Example: "Input.txt". The file must exist		
Post-conditions (success end)	File successfully read and opened to view and edit text content		
Post-conditions (failed end)	File Open failed File not found. I.e. File does not exist		
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Open SRS .txt document for analysis		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Request to Open SRS .txt file	2. Open File browser to select/search for SRS .txt file	Reads and displays Selected File

Use Case Narrative 2(Edit SRS): this use case will enable PROMIRAR to modify SRS document.

Table 4.2: Use Case Narrative for Edit SRS .txt Document

Use Case 2	**Edit SRS Text**		
Goal in content	To Modify/Edit an already opened SRS document or Enter text to be analysed		
Level	This is a basic Edit Text use case		
Parameters	In: Text (Entered Text)		
Pre-conditions	File Opened		
Post-conditions (success end)	Get user keyboard input and update text		
Post-conditions (failed end)			
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Edit/Modify an Opened SRS .txt document		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Request to Edit an Opened SRS .txt file	2. Reads keyboard input and updates SRS document.	Updates SRS .txt document

Use Case Narrative 3(Edit SRS): this use case will enable PROMIRAR to save an edited SRS document.

Table 4.3: Use Case Narrative for Save SRS Text

Use Case 3	**Save SRS Document**		
Goal in content	To Save an Edited/Modified SRS document.		
Level	This is a basic Save Text File use case		
Parameters	In: Text (Entered Text), File Name, Directory		
Preconditions	File/Text Modified, Text Entered, Valid Directory has been selected to save file		
Post-conditions (success end)	File Successfully Saved		
Post-conditions (failed end)	File Save Failed		
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Save a Modified or Edited SRS Text to File		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Request to Save Text to File.	2. Opens File Browser for the user to select a directory and enter preferred name of the file.	Saves Text File to Selected Directory

Use Case Narrative 4 (Output File Directory): this use case will enable PROMIRAR to select a directory to output the IMR document.

Table 4.4: Use Case Narrative for Output File Directory

Use Case 4	**Select Output File Directory**		
Goal in content	To Select A directory to save the .pdf Report file of the analysis		
Level	This is a basic Select a Directory use case		
Parameters	In: Valid Directory Path		
Preconditions	Directory/Path is Valid (i.e. The Inputted Directory Exists)		
Post-conditions (success end)	Directory Successfully Selected		
Post-conditions (failed end)	Invalid Directory Please Select a Valid Directory		
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Select a Directory to save the report of the Analysis		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Request to Select a Directory	2. Opens the Directory Browser for user to select a directory	Reads and Validates the Path of the selected Directory

Use Case Narrative 5 (Ontology Management): this use case will enable PROMIRAR to either select an existing ontology or create one.

Table 4.5: Use Case Narrative for Ontology Management

Use Case 5	**Ontology Management**		
Goal in content	To Select an existing ontology or create one where such does not exist.		
Level	This is a basic Selection use case using combo box		
Parameters	In: Create Ontology / Import Ontology		
Preconditions			
Post-conditions (success end)	Successfully imported Ontology/Created Ontology		
Post-conditions (failed end)	No Ontology imported/Created		
Actor(s)	User (Requirement Engineer)		
Trigger	The user is required to Select an existing ontology or create one otherwise.		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Import/Create an Ontology	2. Accepts/Creates Ontology	Domain Ontology will be available for use for onward analysis.

Use Case Narrative 6 (Select Analysis): this use case will enable PROMIRAR to either select an existing ontology or create one.

Table 4.6: Use Case Narrative for Select Analysis

Use Case 6	**Select Analysis**		
Goal in content	To Select an Analysis to be carried out on the inputted text or opened SRS Document		
Level	This is a basic Selection use case using check boxes		
Parameters	In: Inputted Text / Opened SRS File		
Preconditions			
Post-conditions (success end)	Successfully Selected Analysis		
Post-conditions (failed end)	No Analysis Selected		
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Select one or more Type of analysis to be carried out on the opened file or inputted Text		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Select one analysis type or more from the set of analysis types	2. marks analysis as selected	Inputted Text will be ready for analysis based on selected analysis

Use Case Narrative 7 (Start Analysis): this use case will enable PROMIRAR to execute the selected analysis.

Table 4.7: Use Case Narrative for Start Analysis

Use Case 7	**Start Analysis**		
Goal in content	To Analyse inputted Text or Opened SRS document		
Level	This is where the analysis takes place and further generates reports based on the analysis selected		
Parameters	In: Inputted Text / Opened SRS File, Valid Directory, One or more selected analysis. Out: Analysis Reports		
Preconditions	Directory/Path is Valid (i.e. The Inputted Directory Exists) Inputted Text / Opened SRS File One Analysis or More have been Selected		
Post-conditions (success end)	Analysis Successful Analysis Reports Generated and saved to selected Directory		
Post-conditions (failed end)	Analysis cannot Start (No Analysis Selected, Invalid Directory Inputted, No text Entered/ File Opened to be analysed)		
Actor(s)	User (Requirement Engineer)		
Trigger	User require to Analyse the inputted SRS text or opened SRS document		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Selects to Start Analysis	2. Starts analysis by analysing inputted text based on selected analysis and displays progress to the user during the process. 3. Saves Analysis Report files for each selected analysis to the selected directory	Selected Directory is Updated with analysis report files of the analysis

Use Case Narrative 8 (Help): this use case will enable PROMIRAR to access the help menu.

Table 4.8: Use Case Narrative for Help

Use Case 8	**Help**		
Goal in content	To give the user a Brief description of all the functions of the overall system and how they work for usability reasons		
Level	This is a Basic Help Use Case		
Parameters			
Preconditions	Select a Help Sub Menu		
Post-conditions (success end)	View Selected Help Text		
Post-conditions (failed end)			
Actor(s)	User (Requirement Engineer)		
Trigger	User is required to understand how the system can be used or how the different analysis types work, i.e. the algorithm of the analysis types		
Description (event flow)	Actor action	System respond	Affected data object with operation
	1. Selects A Help Sub Menu	2. Displays A Description of the selected Menu	

4.3.2 Class Diagram for PROMIRAR

This depicts a static view of the classes or instances in the model. The combined class diagram for PROMIRAR models the data elements in the system, the ways in which data may be grouped together, and the association between them. The attributes associated with each class are also identified. The class diagram is shown in Figure 4.2 and the relationship between the classes and their functionalities is as shown in Table 4.9.

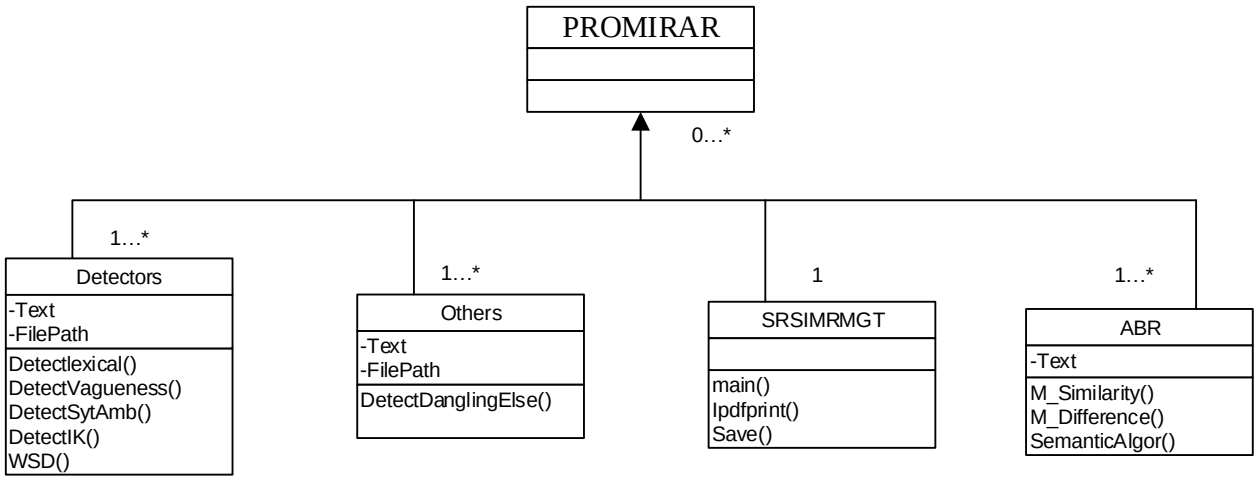


Figure 4.2: Combined Class diagram for PROMIRAR

Table 4.9: Table showing the Architecture's Classes and their Functionalities

Class	**Functions**	**Description**
Detectors	Detectlexical() DetectVagueness() DetectSytAmb() DetectIK() WSD()	This class contains all the functions required to identify a requirement document containing IMR.
SRSIMRMGT	Main() Ipdfprint() Save()	This class contains the main function that executes every other class as well as calling the IPDF to print the identified IMR and then Save function to save the IMR.
ABR	M_Similarity() M_Difference() SemanticAlgor()	This class contains functions to do the ABR processing of finding similarity and difference computation of requirements documents as well as execute the semantic matching algorithm.
Others	DetectDanglingElse()	This class is put in place for other ambiguity functionalities such as *danglingelse* ambiguity.

4.3.3 Activity Diagram for IMR Classification

The sequence of flow of some of the activities within PROMIRAR was also modelled using the activity diagram. Figure 4.3 depicts the activity diagram for IMR classification in PROMIRAR when a software requirements specification document is inputted in PROMIRAR.

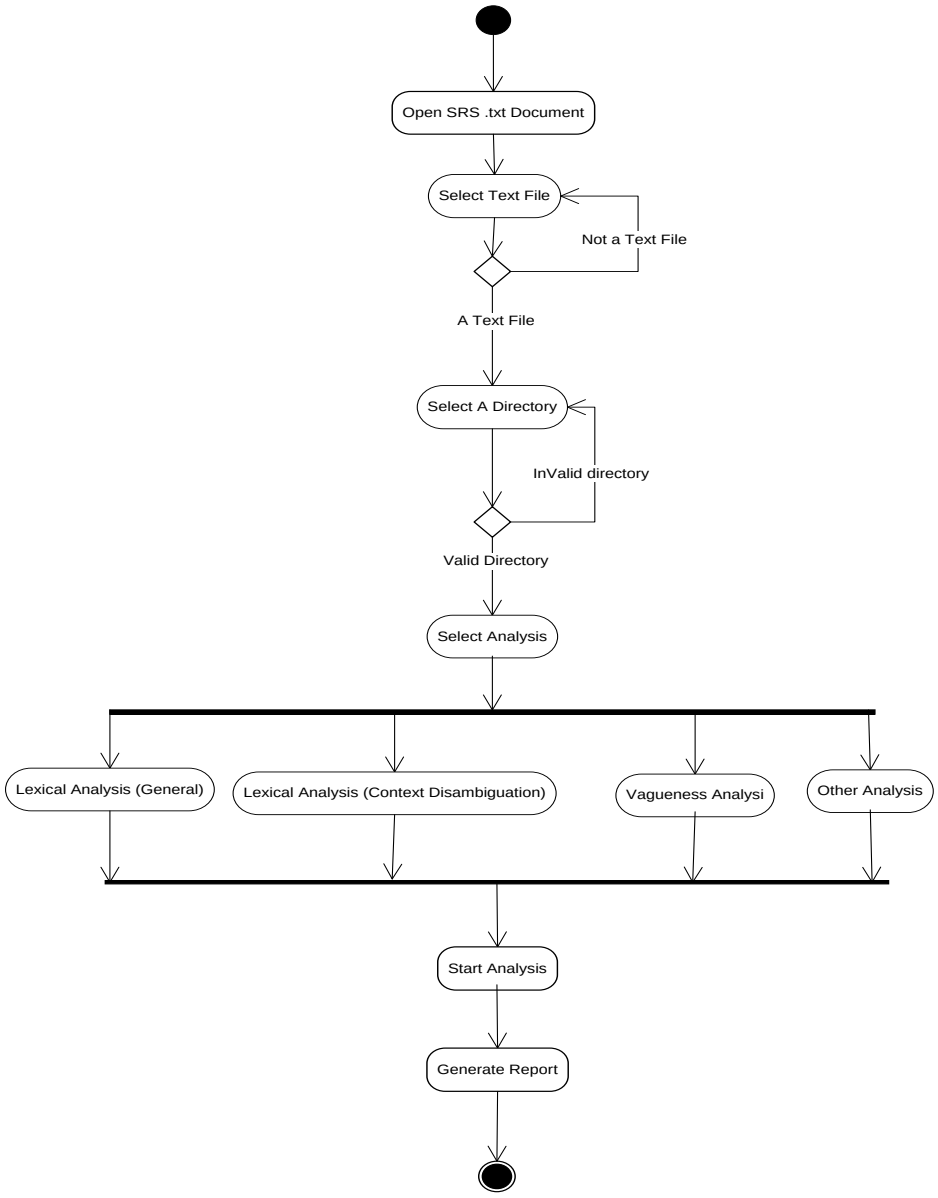


Figure 4.3: Activity Diagram for IMR Identification in PROMIRAR

4.3.4 Sequence diagram for IMR Identification in PROMIRAR

Sequence diagram depicts the interaction of messages between the system and users in time sequence. Figure 4.4 shows the sequence diagram for identifying implicit requirements in PROMIRAR.

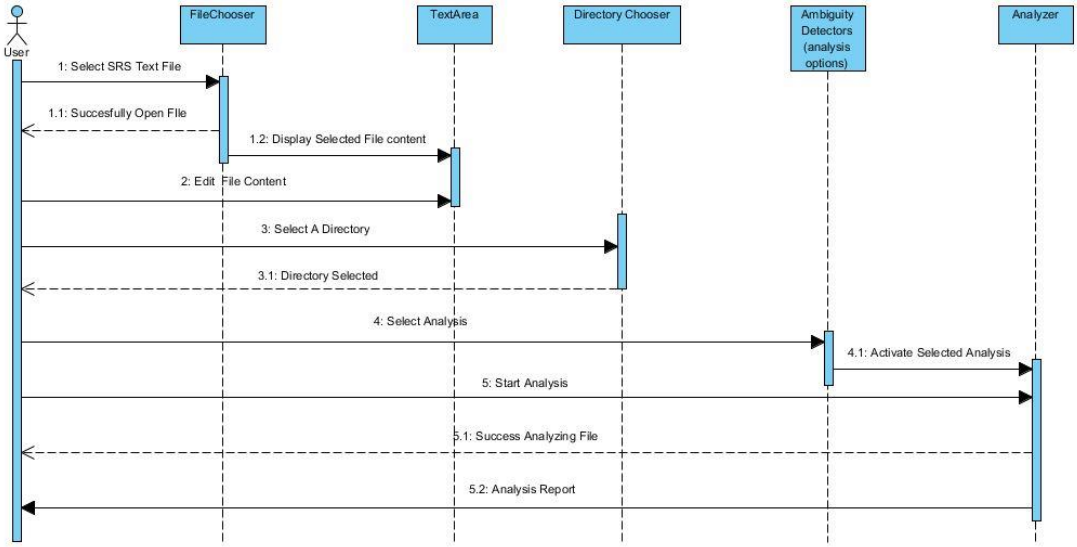


Figure 4.4: Sequence Diagram for IMR Identification in PROMIRAR

4.3.5 Workflow of PROMIRAR's Operation

The flowchart of PROMIRAR's workflow starting from the point where the user inputs the software requirements specification document to the output of the implicit requirements detected is as shown in Figure 4.5.

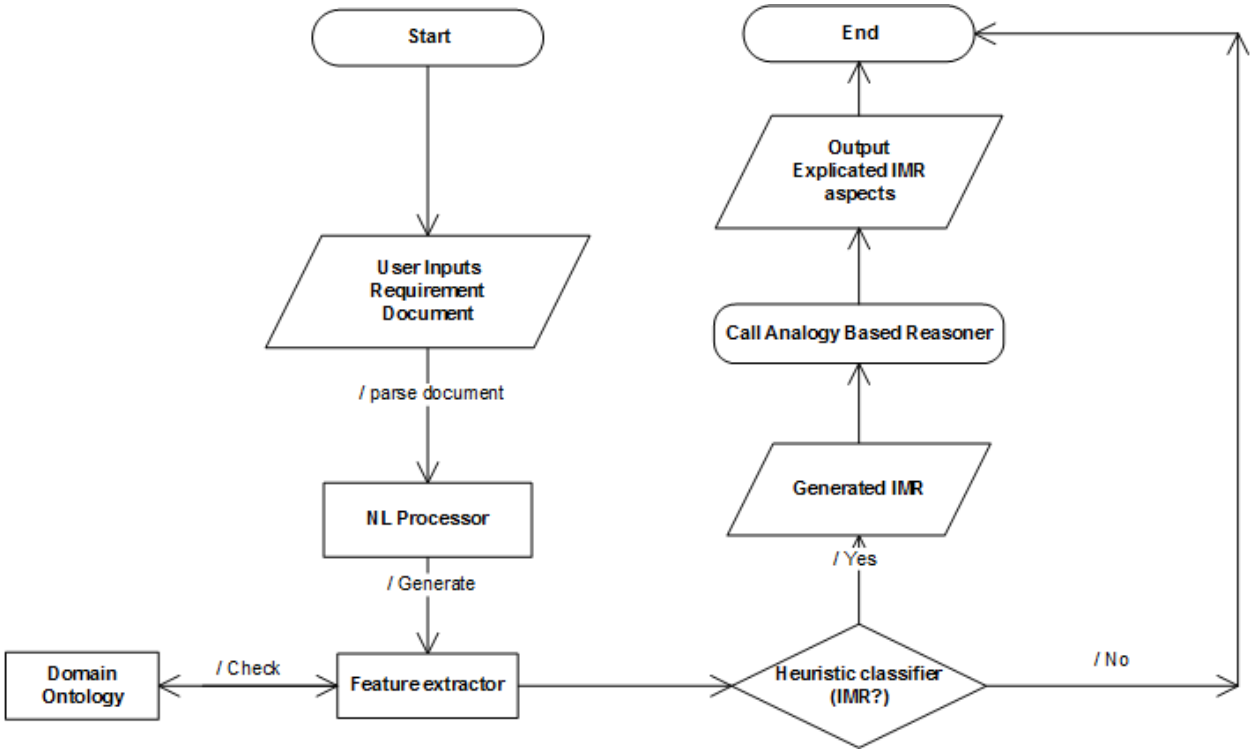


Figure 4.5: Flowchart Diagram for PROMIRAR'S Workflow

4.4 FILE DESIGNS

The File System used in PROMIRAR is divided into 3, which includes: The Input Files, The Output Files and The Corpus Files.

4.4.1 Input Files

Input Files are text documents, they are the SRS documents to be analysed by the system. They contain a natural language representation of the requirements to be analysed by the system for ambiguities.

The System is designed to open only files/documents in .txt, .doc, .pdf formats. All formats are converted and processed in .txt format. This is because system requirements are represented in natural language, and the use of text files helps to achieve this goal. Text files only give room for natural language representation (plain text) and eliminate the issue of us having diagrams, models, tables, etc. in the requirement document to be analysed.

4.4.2 Output/Report Files

The Output files are ".pdf" files which are stored in the directory selected by the user. The output files are stored with different filenames which are made up of the type of analysis which the file is reporting and the name of the analysed text file. This file basically documents the analysis results during the analysis and can be viewed by opening the selected directory after the analysis is completed or automatically opened after the analysis is finished by activating the auto view option in the system menu.

The Report files are generated based on the analysis selected by the user before starting the analysis. Each selected analysis has its own report file. The report file basically rewrites the analysed text document and highlights using colours and font type to show detected ambiguous words and phrases in the text document, the report file also contains a percentage figure of the detected ambiguities for the selected analysis.

4.4.3 Corpus Files

Corpus Files are read-only text files stored in the corpus folder/directory, the corpus directory serves as a database containing several files where each file stores keywords and phrases that are tagged as "potentially IMR" for a particular type of IMR, these keywords were gotten from literature reviews. The list of this keywords is in exhaustive and more keywords can be added as deemed fit by the requirements engineer.

During analysis, the text document checks the corpus directory for the text file that is tied to the selected analysis and fetches all the keywords contained in the selected analysis corpus file and scans the text document to be analysed if any of those keywords exists. Not all analysis requires the corpus to carry out analysis.

4.5 ONTOLOGY DESIGN

This section gives the design of the ontology used in this research. Part of a Course Management System (CMS) ontology is used to describe the design process.

4.5.1 Domain Requirements

Some of the ontology's requirements were defined using the following competency questions:

i. What is the minimum number of course units to register for in a session?
ii. What courses are to be taken in each of the levels?
iii. What is the procedure for registration?
iv. What necessary information does a student need from his/her level adviser?

4.5.2 Conceptual Modelling of the Domain

The following steps were taken in building the course registration ontology for use with the Course registration requirements specification document:

i) Define classes in the ontology,
ii) arrange the classes in a taxonomic (subclass–superclass) hierarchy
iii) define slots and describing allowed values for these slots,
iv) fill in the values for slots for instances,
v) Define relationships among the various classes.

Protégé 4.1 (an ontology editor) was used to model the class hierarchy.

The Entities of the domain and their subclasses are shown below:

i. Student
    a. Matriculation Number
       This was made a functional requirement since no two students can have the same matriculation number
ii. Course Registration
iii. Get Form Online
iv. Pay Fees
v. Get Manual Form
vi. See Level Adviser

a. 100 Level
b. 200 Level
c. 300 Level
d. 400 Level
e. Maximum Unit
f. Minimum Unit
g. Advice
h. ReRuns
i. Course Codes and Titles

vii. Fill Form
viii. Submit Form

The Relationships between Entities are:

i. A STUDENT IS_IN a LEVEL
ii. A STUDENT OFFERS some COURSES
iii. Some COURSES are OFFERED in a LEVEL
iv. Some COURSES have PREREQUISITES in Some LEVELS
v. Some COURSES are being OFFERED BY a STUDENT (This is an inverse functional requirement to "A STUDENT OFFERS some COURSES"
vi. COURSE_CODE HAS TITLE
vii. COURSE_CODE HAS UNIT

A cross section of some sample screenshots of the various ontographs of the CMS domain ontology such as the course registration, course advisor, and 100 level courses is as shown in Figure 4.6-4.8.

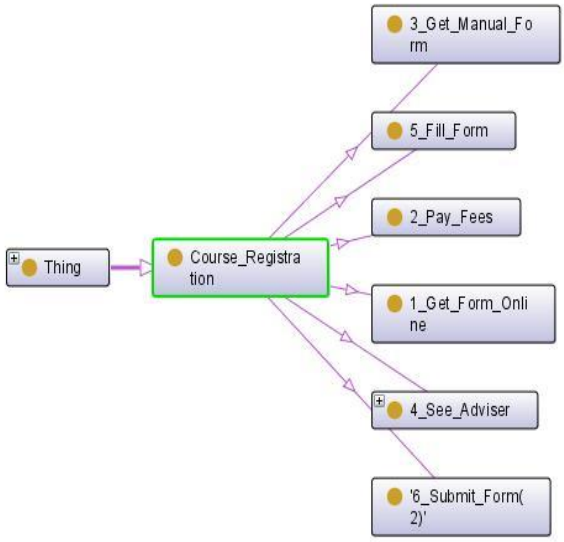


Figure 4.6: An Ontograph of the Various Steps Needed for Registration

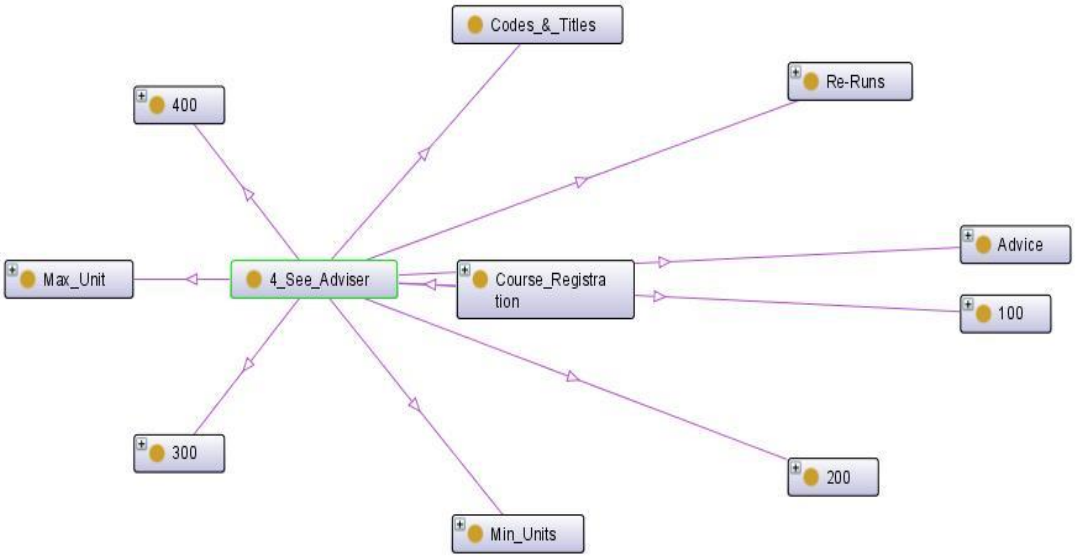


Figure 4.7: An Ontograph of the basic things to get from a Level Adviser

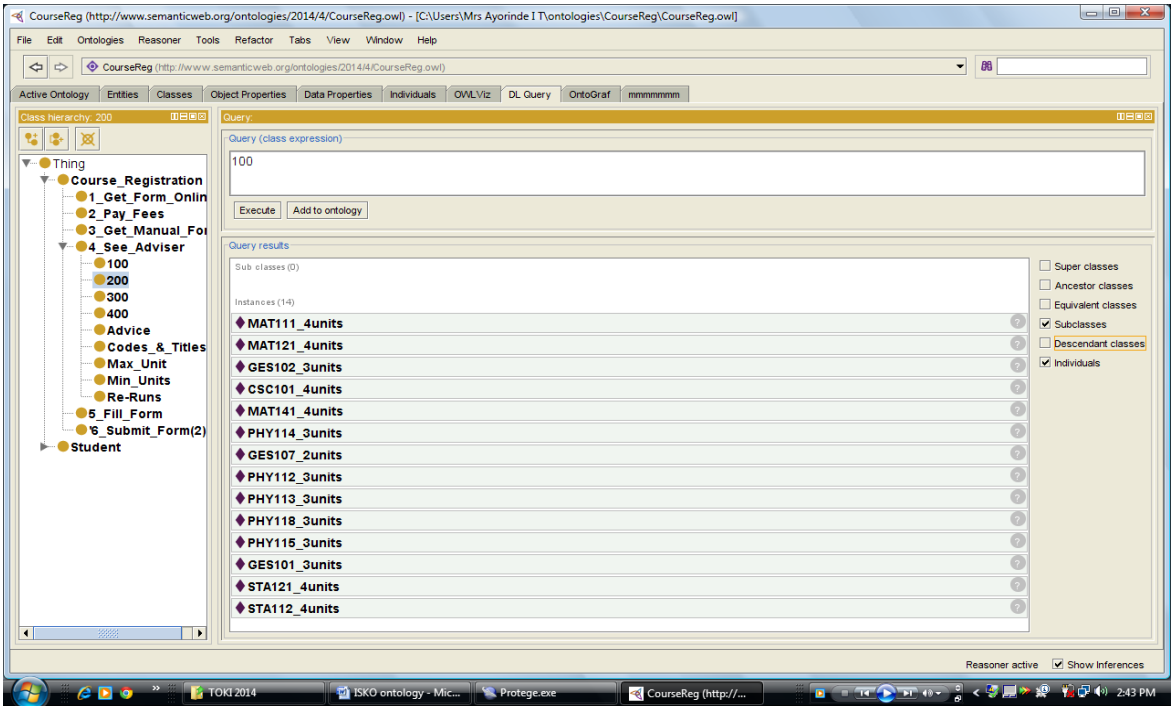


Figure 4.8: Result of a Query of 100 Level Courses

4.6 ALGORITHM DESIGN

The algorithms implemented in the PROMIRAR tool are developed from the knowledge gotten from reviewing relevant literature to understand different ambiguity types and further developing algorithms for detecting each type. The following types of ambiguity where considered in the implementation: lexical ambiguity, vagueness, incomplete knowledge and others (e.g. Dangling Else, Ambiguous Variables, Implicit Cases, etc.).

4.6.1 Lexical Ambiguity

According to Berry *et al*., (2003), lexical ambiguity occurs when a word can have more than one meaning, and this can be further divided into two which includes homonym and polysemy.

I. Homonym

This is when two different words have the same written or phonetic representation. For example, the word “bank” can mean a financial institution, or a sloping land beside a body of water.

II. Polysemy

This occurs when a word has more than one related meanings but a single etymology.

From the above understanding of lexical ambiguity, an algorithm to automatically detect if a word in a sentence is lexically ambiguous or not was developed, and this was achieved by using the Wordnet dictionary to check if a word has more than one meaning in the context (Part of Speech) in which it was used in the sentence.

```
1.  Identify all sentences in the Document
2.  Break identified Sentences into words
3.  Identify the part of speech of each word
4.  Check the dictionary for the meaning of each word
    in the sentence based on the identified part of
    speech
5.  If the word exist in the dictionary count the
    available meanings of the word based on the part
    of speech
6.  If the available meaning is >1 then the word is
    ambiguous
7.  Else if the available meaning is <= 1 then mark
    the word as not ambiguous (available meaning <1
    means the word does not exist in the dictionary).
```

Figure 4.9: Lexical Analysis Algorithm for PROMIRAR

The algorithm in Figure 4.9 ignores the fact that a word that has more than one meaning in the dictionary might not be ambiguous in the context in which it is used in the document, in the sense that a word with more than one meaning can be used in different sentences in the document but referring to the same meaning in the dictionary, the above algorithm do not handle this scenario as it just concludes that a word with more than one meaning is lexically ambiguous. This brings us to the concept of word sense disambiguation.

Word Sense Disambiguation (WSD) is the act of finding the actual meaning that matches the context in which a word that has more than one meaning was used in a sentence. For example in the sentence “I have an interest in arts”, the word “interest” is an ambiguous word in the sense that it may mean: appreciation, or a charge for borrowing money. For humans it is easy to tell that interest in that context is talking about appreciation and not otherwise but it is not so for computers (Banerjee, 2005). In this project, the GANNU WSD[2] was used as was used for lexical ambiguity detection called “Lexical Analysis – context disambiguation”. This algorithm tends to carry out some degree of disambiguation using the part of speech and the number of occurrence of words marked as ambiguous by the algorithm in Figure 4.10.

1. Identify all sentences in the document
2. Break identified sentences into words
3. Identify the part of speech of each word
4. Check the dictionary for the meaning of each word in the sentence based on the identified part of speech.
5. If the word exist in the dictionary count the available meanings of the word based on the part of speech
6. If the available meaning is <=1 then we say the word is not ambiguous (available meaning <1 means the word does not exist in the dictionary).
7. If the available meaning is >1 check if the word appears again in the document
8. If the word does not appear again in the document then we say the word is not ambiguous
9. If the word appears again in the document compare the part of speech of the next appearance of the word with that of initial appearance of the word
10. If the part of speech is the same we say the word is not ambiguous
11. Else if the part of speech is different then we say the word is ambiguous
12. Go back to step 9 until all word appearance in the document is treated

Figure 4.10: Lexical Analysis –Context Disambiguation Algorithm

[2] https://sourceforge.net/p/gannu/wiki/WordSenseDisambiguation/

The algorithm in Figure 4.10 puts a form of control in the lexical ambiguity detection process by minding the context in which the word is used in the document and says a word is not ambiguous in a document unless it is used more than once in the document with differences in the context (part of speech).

4.6.2 Vagueness

Another ambiguity type treated in this project is Vagueness, Vagueness occurs when a phrase has a single meaning from the grammatical point of view, but still leaves room for interpretation, when considered as a requirement. For example in the sentence "The system should react as fast as possible" the word "fast" is a vague word in the sense that it leaves room for us to further define how fast the system should be in carrying out its operations (Gleich, 2010).

According to research, it is observed that most words referred to as vague in the ambiguity handbook (Berry *et al*., 2003) are adjectives and adverbs this lead to the conclusion that all adverbs and adjectives are potentially vague and should be treated as such (Gleich, 2010).

From the above understanding gotten from reviewing relevant literature, an algorithm was developed to automate the detection of vagueness in an SRS document. The algorithm is as shown in Figure 4.11:

```
1.    Identify all sentences in the document
2.    Break identified sentences into words
3.    Identify the part of speech of each word
4.    If the part of speech is == ADVERB or part of
      speech == ADJECTIVE then mark the word as vague
5.    Else mark the word as not vague
```

Figure 4.11: Vagueness Analysis Algorithm

4.6.3 Other Ambiguities

The term “Other Ambiguities” was coined mainly in this project. “Other ambiguities” do not refer to a special type of ambiguity but refers to the detection of ambiguities in an SRS document by making use of the corpus.

A corpus or text corpus can be defined as a large and structured set of texts (nowadays usually electronically stored and processed). They are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific language territory.

In this project, the corpus refers to a directory, which contains text files where each text file contains a list of keywords and phrases referred to as potentially ambiguous to a specific type of ambiguity.

The keywords or phrases in each file are not randomly generated but gotten from research documents such as the Ambiguity Handbook (Berry, 2003). A linear search algorithm was implemented for searching the corpus text files for keywords and phrases. The algorithm is as shown in Figure 4.12.

```
1.    Identify all sentences in the document
2.    Break identified sentences into words
3.    Read the Corpus text file of the Ambiguity Type
4.    Get the first word
5.    Compare the word to each line of the corpus text
      file
6.    If match found mark the word as ambiguous and go
      to next word
7.    Else mark the word as not ambiguous and move to
      next word
8.    Go back to step 5 until all words in the document
      is compared to each line of the corpus text file.
```

Figure 4.12: Corpus Search Algorithm

4.7 IMPLEMENTATION LANGUAGE AND PLATFORM

The Eclipse Platform was chosen as the implementation platform. It is an open source tool that is designed for building Java applications. The Eclipse Platform's principal role is to provide the tool providers with mechanisms to use and rules to follow that lead to seamlessly integrated tools. It also provides useful building blocks and frameworks facilitating the development of new tools. Eclipse operates under an open source paradigm, with a common public license that provides royalty free source code and worldwide redistribution rights for tool developers with flexibility and control over their software technology.

Eclipse-based tools give developers freedom of choice in a multi-language, multiplatform, multivendor environment. Eclipse provides a plug-in based framework that makes it easier to create, integrate and utilise software tools. The Eclipse Platform is written in the Java programming language and comes with extensive plug-in construction toolkits and examples.

The Eclipse Platform is designed to meet the following requirements:

i. Support the construction of a variety of tools for application development
ii. Support an unrestricted set of tool providers, including independent software vendors (ISVs)
iii. Support tools to manipulate arbitrary content types (HTML, Java, C, JSP, etc.)
iv. Facilitate seamless integration of tools within and across different content types and tool providers
v. Run on a wide range of operating systems, including Windows and Linux
vi. Capitalise on the popularity of the Java programming language for writing tools

4.8 IMPLEMENTATION OF PROMIRAR

This section states the modules of PROMIRAR and further gives a detailed description of each of the modules and how they are implemented. Each of the modules is integrated together to make the whole system. The modules are as follows:

a. File Module
b. Edit Module
c. Select Directory Module
d. Select Ontology Module
e. Select Analysis Module
f. Analysis Module
g. View Module
h. Help Module

A view of the main screen of a PROMIRAR is shown in Figure 4.13.

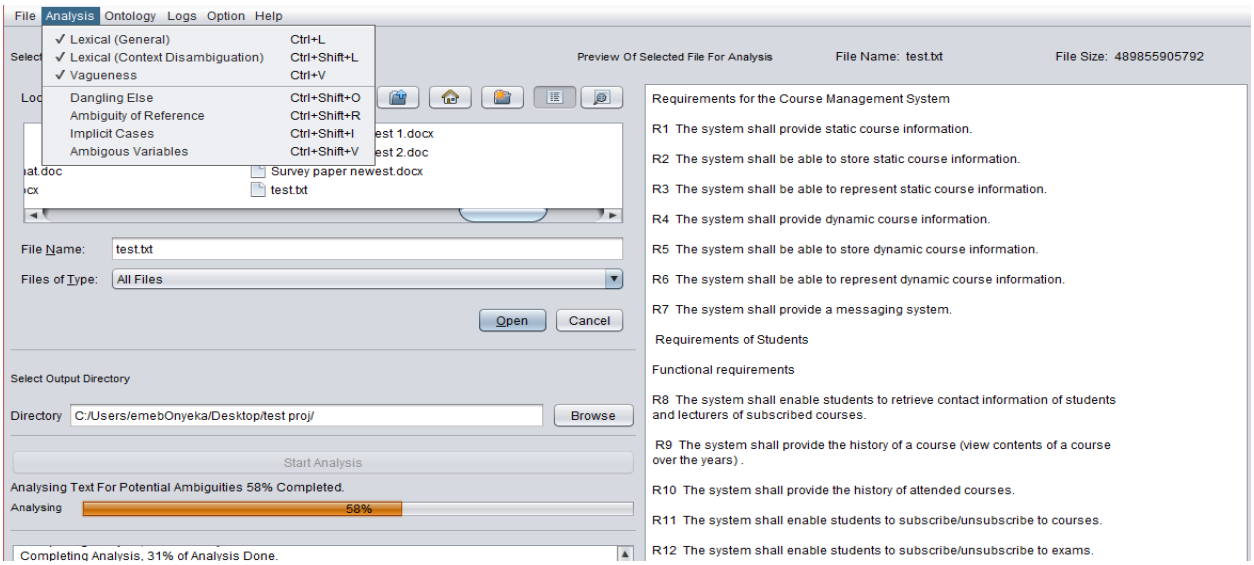


Figure 4.13: A Snapshot of PROMIRAR Main Screen

A description of the APIs used in this module and their functions in this module:

i. Wordnet Java API

WordNet is a large lexical database of English, developed under the direction of George A. Miller. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms, each expressing a distinct concept.

This API was used to get the meaning of words and number of senses/meaning of words in the analysed text file.

ii. Apache Open NLP

Apache OpenNLP library is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution. This API was used to achieve the sentence detection process, breaking the sentence into words, and getting the part of speech of the words.

iii. IText PDF API

IText is a PDF library that is used to CREATE, ADAPT, INSPECT and MAINTAIN documents in the Portable Document Format (PDF). This API was used to create and write into PDF files (Analysis Reports) during the analysis process.

iv. Gannu API

Gannu is a Java API, command line and graphical tools for performing AI tasks such as Word Sense Disambiguation.

The Gannu WSD module used in this thesis allowed for the following: i) Perform comparisons between different Bag of Words Model WSD systems; ii) Disambiguate RAW text

v. Protege-OWL API

The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning based on Description Logic engines.

4.9 DESCRIPTION OF PROMIRAR VIEWS

A description of the various views of PROMIRAR is as explained below while Figure 4.14 shows the highlighted part and numbers representing each interface goal.

4.9.1 The Main PROMIRAR Window

i. **Select Analysis:** The highlighted part with tag number "1" provides the user with all the available analysis types in the system where the user is allowed to select from the list of items (Analysis Types) by checking the box associated with the analysis type.

ii. **Open-File:** The highlighted part with tag number "2" provides the user with a file chooser, which allows the user to select a text file by browsing the system directories.

iii. **Select Directory:** The highlighted part with tag number "3" provides the user with a directory chooser, which allows the user to select a directory to save the output file.

iv. **Start Analysis:** The highlighted part with tag number "4" provides the user with a Start Button, which initiates the analysis after the file has been opened and a valid directory has been selected and also provides the user with a progress bar which shows the progress of the analysis in percentage.

v. **Edit Text:** The highlighted part with tag number "5" provides the user with a Text area, which outputs the content of the selected text file and also enables the user to be able to modify the content of the text file before starting the analysis.

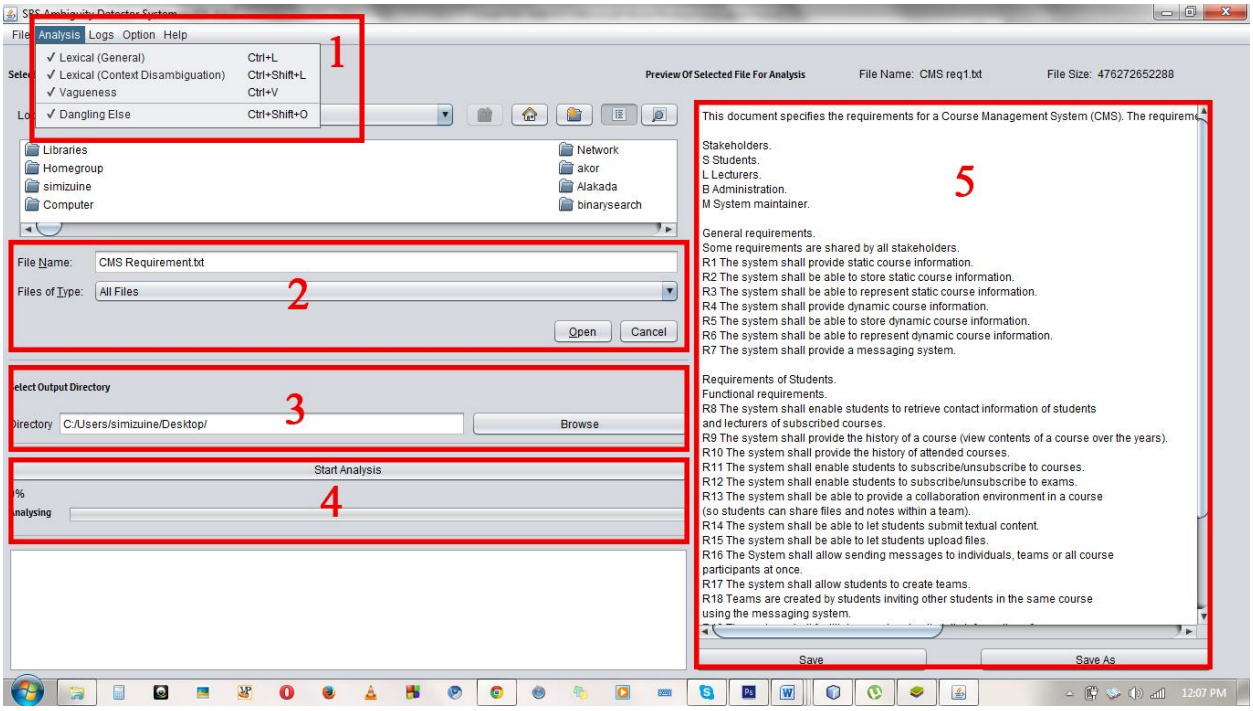


Figure 4.14: PROMIRAR Main Screen with Highlight of Interface of Major Goals

4.9.2 The About PROMIRAR Window

This window displays information about the PROMIRAR in an inactive text area. The windows are accessed by clicking on the Help Menu >> About PROMIRAR menu item or by using the shortcut “Ctrl + Shift + A”. The screen shot of about PROMIRAR window is as shown in Figure 4.15.

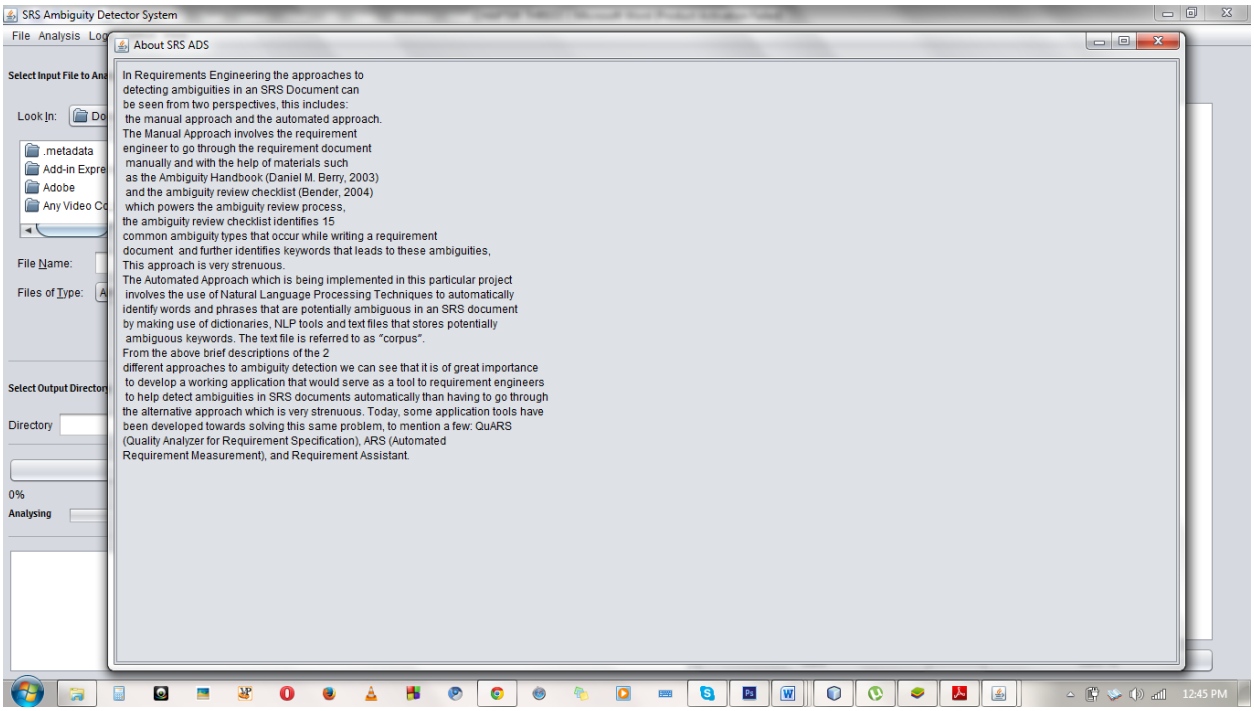


Figure 4.15: PROMIRAR Help Window

4.9.3 The About Analysis Window

This window displays information about PROMIRAR Analysis types (see Figure 4.16) and how they work i.e. the algorithm. This was achieved by using a window, which contains multiple tabs where each tab carries information about a particular type of analysis provided by the system.

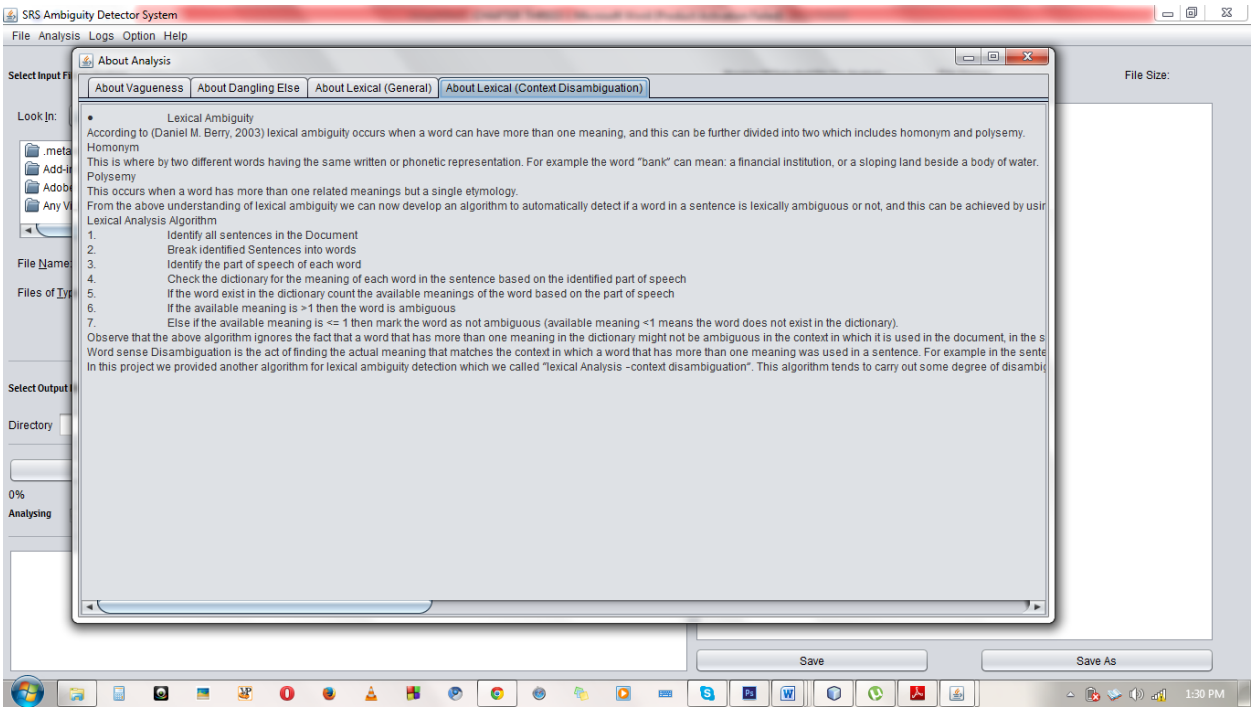


Figure 4.16: PROMIRAR Help Window (About Analysis)

4.9.4 Report Generation in PROMIRAR

After the SRS text file has been opened and a valid directory has been selected and modification has been made to the file if necessary then the analysis can be started, once the analysis is done (i.e. when the progress bar gets to 100%) reports are generated and stored in the selected directory, this report can be set to open automatically once the analysis is complete by activating the check box menu item "Auto Open Report" under the option menu as shown in Figure 4.17.

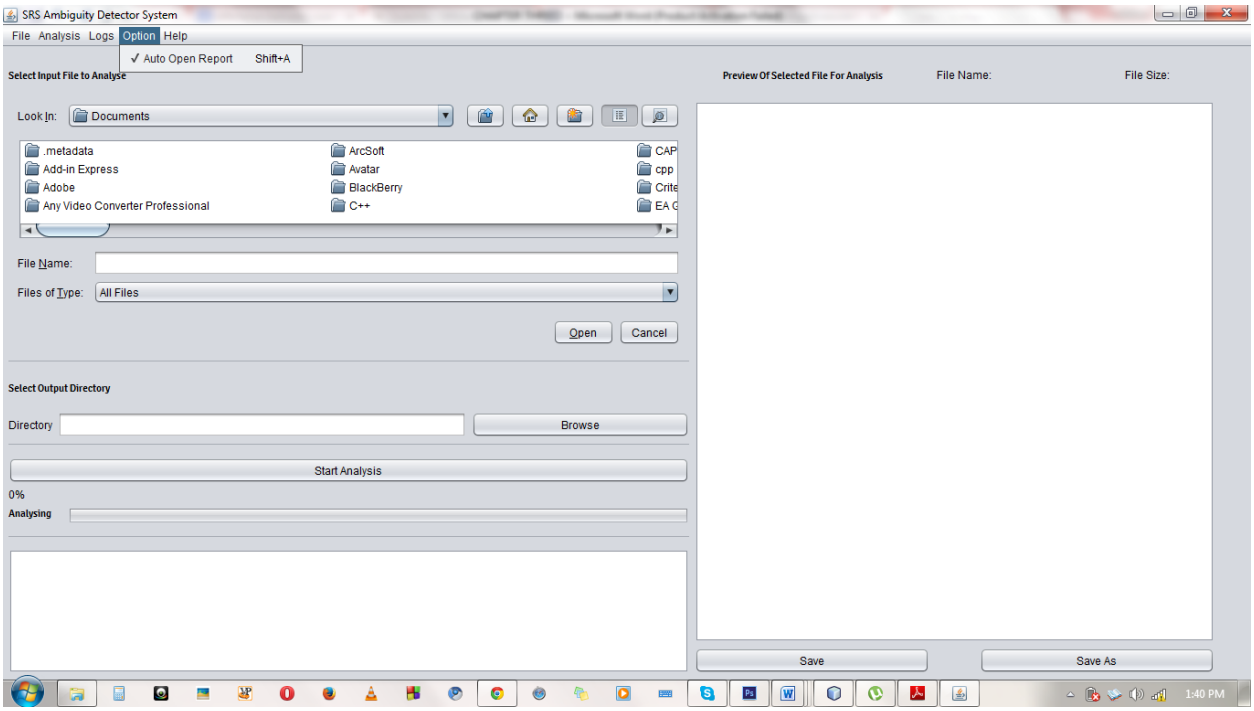


Figure 4.17: PROMIRAR Main Screen Activating the "Auto view Report" Option

The reports generated are "pdf" files, so the software requires the system user to have any version of Adobe Reader application installed on the system to be able to view the reports. The content shows the type of IMR analysis done, which entails the content of the analysed SRS document, highlighting the detected IMR by using colours and italic font as well as its explicated requirements are shown in Figure 4.18-4.19.

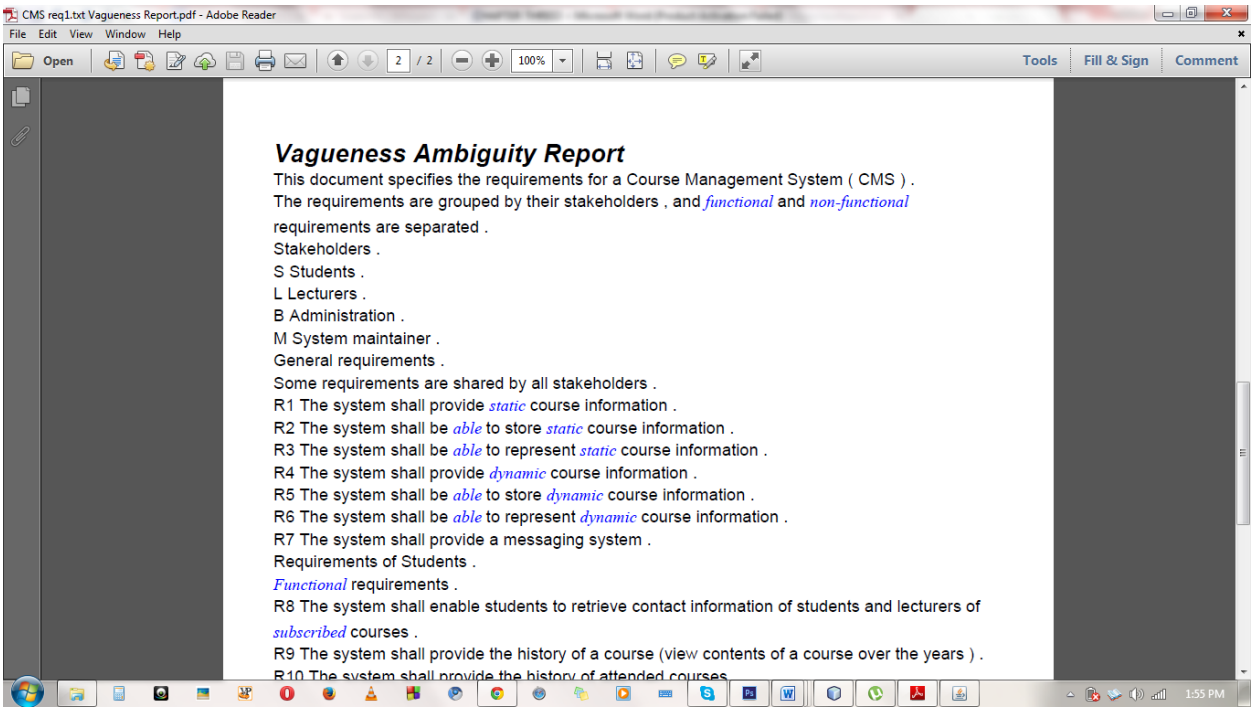


Figure 4.18: Screen of PROMIRAR Vagueness Report File

Lexical Ambiguity Report

The C&C shall *provide* the users with real-time data regarding the measured values , as collected from the *various* sensors *part* of the *network* .
The C&C shall *support* the *configuration* of ranges for sensor readings (maximum and minimum allowed values) .
The C&C shall *report potential* sensor malfunctions to the users , when the *reading* is " *Suspicious* " or " Invalid " .
The C&C shall *allow* users to *validate* readings that were qualified as " *Suspicious* " or " Invalid " .
This means that users shall *be* allowed to *qualify* as " Good" , sensor readings that were classified as " *Suspicious* " or " Invalid " automatically .
The C&C shall notify users if there are manually modified values , whenever it presents sensor data to them .
The C&C shall *have* the sensor readings displayed in a GIS *environment* .
The C&C shall *represent* the sensor nodes in the *system* as two-dimensional sets of dots (or symbols) in a *rectangular* panel .
The C&C shall *provide* a visual *display* of sensor readings to the users , by clicking on each sensor node 's *representation* .
The C&C shall *allow* for the visual *selection* of elements of *interest* by using layers of *information* .
Each *layer* shall *be* associated with a *particular type* of *element* of *interest* .
The C&C shall *set* the *appropriate* endangerment *level* , according to the sensor readings .
The C&C shall *provide* the users with *historical* data regarding the measured values .
The C&C shall *keep* a *history* of collected sensor readings of up to 1 *year* .

Figure 4.19: Screen of PROMIRAR Lexical Ambiguity Report File

4.10 SUMMARY

In this chapter, the full scope of the design and implementation of PROMIRAR for identification and management of implicit requirements has been discussed. The various UML diagrams (Class Diagram, Use Case diagram, Activity diagram, etc) which make up the design of PROMIRAR was reported. The various ontology design, algorithm design (Lexical, Vagueness and other ambiguities), API's (Wordnet, Apache Open NLP, IText PDF, Gannu, Protégé-OWL) used to develop each module, language and platform for developing PROMIRAR was discussed. The chapter concludes with the various screenshots and descriptions showing the various interfaces and outputs of PROMIRAR.

CHAPTER FIVE

EVALUATION

5.1 INTRODUCTION

This chapter gives a report of the evaluation of PROMIRAR tool and the process framework. Two kinds of evaluation were performed. Firstly, an evaluation of the performance of the tool using three (3) different requirements specification documents in three different experiments was conducted. Secondly, an evaluation of the application of the tool as a support for the process of handling implicit requirements within software organisations was conducted using two (2) software development organisation as a case study.

5.2 PERFORMANCE EVALUATION OF PROMIRAR

A performance evaluation of PROMIRAR was conducted using three (3) different requirements specification document in three different experiments. This approach was adopted so as to assess the quality of the detected IMR.

5.2.1 Overview of Source Requirements Documents

The following requirements specifications were used for the evaluation as discussed below. These requirements are standard requirements documents that are open and available online.

i. **Course Management System Requirements specification**: The Course Management System (CMS) requirements (see Figure 5.1) as used in Abma (2009) describes some basic functionality like course enrollment, course lecture material and timetable upload, students grading and e-mails notification to students. The requirements document contains sixteen requirements as artefacts and seventeen relations that connect them.

ii. **Smart City (EMbedded MOnitoring)**: The EMMON project (EMMON, 2010) is a European Research and Development (R&D) project. The motivation for EMMON originated from the increasing societal interest and vision for smart locations and ambient intelligent environments (smart cities, smart homes, smart public spaces, smart forests, etc). The development of embedded technology allows for smart environments creation and scalable digital services that increase the human quality of life.

iii. **Tactical Control System (TCS) requirements**: This project (Naval, 2000) was designed for the Naval Surface Warfare Center-Dahlgren Division and Joint Technology Center/System Integration Laboratory, Research Development and Engineering Center, U.S. Each of the requirements is as shown in Appendix A.

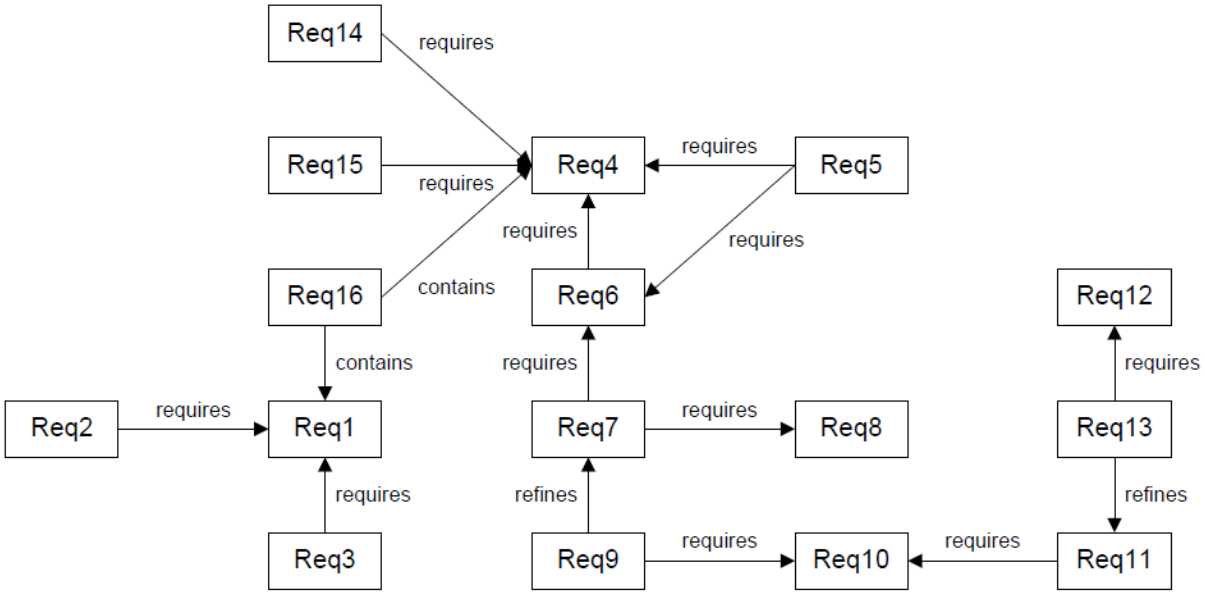


Figure 5.1: Structure of Relations of CMS Requirements
Source: Abma (2009)

5.2.2 Background of the Subjects

Eight (8) subjects were used to conduct three different experiments with each of the requirements documents. The background of the subjects is as follows (see Table 5.1). The subjects include:

i. 2 software engineering experts with industrial experience,
ii. 2 software engineering master students at the Montclair State University (MSU) USA.
iii. 2 PhD students and 2 Faculty members at the MSU. All of who are doing research in software engineering.

Table 5.1: Subjects' Profession and Experience Index

#Subjects	Profession	Organization	Experience
2	Software Engineer	Software Engineering Assoc.	10-15 years.
2	Masters Student	MSU(Software Engr. Major)	0-5yrs.
2	PhD Student	MSU(Software Engr. Major)	0-5yrs.
2	Faculty	MSU(Researcher + Industry experience)	6-10 yrs.

5.2.3 Description of Experimental Procedure

In each case, they were asked to mark implicitness in the sample IMR identification form as shown in Table 5.2 and also make use of the PROMIRAR tool. The subjects who are a group of computing professionals, comprising software developers, academics and research students are skilled in ambiguity detection and were further provided with the Ambiguity Handbook (Berry *et al*., 2003), and as well were trained on identifying implicit requirements. They were given the following instructions:

i. For each specified requirement, mark each requirement based on its implicit nature noting that a requirement may contain more than one form of implicitness.
ii. Secondly, for each requirement specify the degree of criticality of each implicitness on a scale of 1 to 5. 1 being least critical to 5 being most critical.

The defined sources of implicitness include:

i) Ambiguity (**A**);
ii) Incomplete Knowledge (**IK**);
iii) Vagueness (**V**); and
iv) Others

Table 5.2: Sample IMR Identification Form

S/N	Requirement	Type of Implicitness		Criticality				
1	The system shall provide a password reset function, which resets the password and emails it to the user		(A)	1	2	3	4	5
			(IK)	1	2	3	4	5
			(V)	1	2	3	4	5
			(O)	1	2	3	4	5
2	The system shall facilitate searches within all dynamic information and files in a course		(A)	1	2	3	4	5
			(IK)	1	2	3	4	5
			(V)	1	2	3	4	5
			(O)	1	2	3	4	5
3	The system shall enable students to subscribe/unsubscribe to courses							

5.2.4 Metrics for the Performance Evaluation

Precision and Recall are the two main evaluation metrics used in information retrieval system. The measures were defined in Sanderson (2010). Generally, these measures are evaluated using the actual values as against the predicted outcome as shown in Figure 5.2. Where:

Precision is the fraction of retrieved documents that are relevant to the query.

$$\text{Precision} = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ document\}|} \tag{5.1}$$

$$Precision = \frac{TP}{TP+FP} \tag{5.2}$$

Recall is the fraction of the documents that are relevant to the query that is successfully retrieved.

$$\text{Recall} = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ document\}|} \tag{5.3}$$

$$Recall = \frac{TP}{(TP+FN)} \tag{5.4}$$

F-Measure is the harmonic mean of precision and recall.

$$F - Measure = \frac{Precision * Recall}{Precision + Recall} \quad (5.5)$$

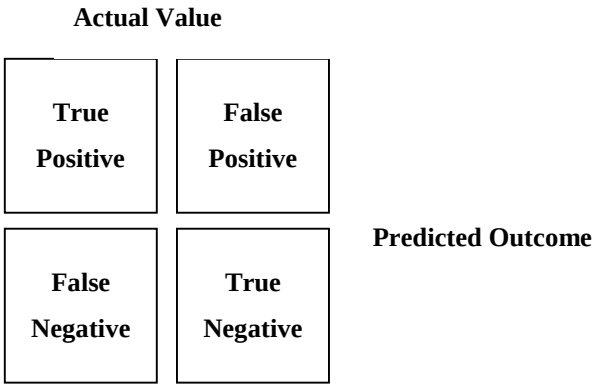


Figure 5.2: Performance Evaluation (Actual Value vs. Predicted Outcome)

Where:

True Positive (TP): The true label of the given instance is positive, and the classifier also predicts it as a positive.

True Negative (TN): The true label is negative, and the classifier also predicts a negative.

False Positive (FP): The true label is negative, but the classifier incorrectly predicts it as positive.

False Negative (FN): The true label is positive, but the classifier incorrectly predicts it as negative.

In the context of this thesis, the performance evaluation was conducted using manual human experts' performance measure against that of PROMIRAR as shown in Figure 5.3. The performances were measured in terms of precision (P), recall (R) and F-measure (F) defined as follows:

i. **Precision**: It shows the percentage of IMR judged by experts that were also retrieved by the tool.

ii. **Recall**: It shows the percentage of IMR judged by experts in the set of IMR retrieved by the tool.

iii. **F-Score**: It is the harmonic mean of Precision and Recall.

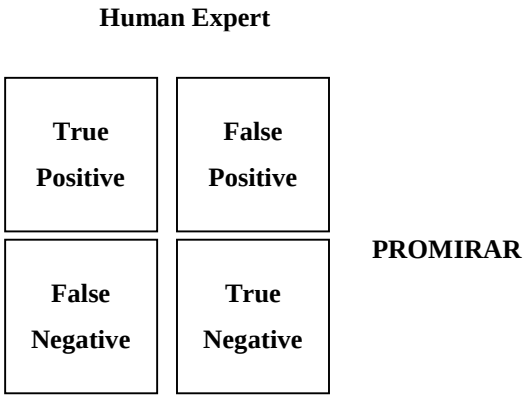


Figure 5.3: Performance Evaluation (Human Expert vs. PROMIRAR)

Where:

a. TP (True Positives) is the number of correctly identified IMR by both expert and PROMIRAR
b. FN (False Negatives) is the number of IMR judged by expert as correct but identified by PROMIRAR as not IMR,
c. FP (False Positives) is the number of Requirements judged by experts as non-IMR but was classified by PROMIRAR as IMR,
d. TN (True Negative) is the number of correctly identified non-IMR by both expert and PROMIRAR.

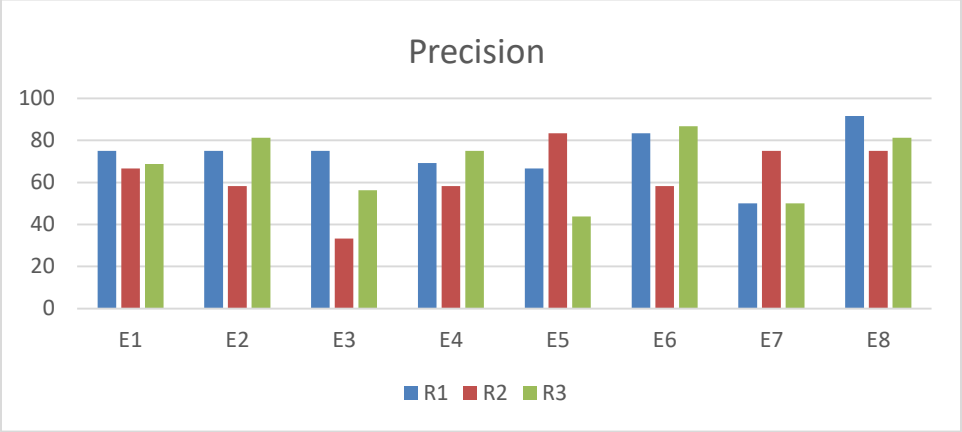
5.2.5 Performance Evaluation Results

For each of the requirements document given in Section 5.2.1, each requirements document was coded RS1 to RS3 (i.e. R1: CMS Requirements Document, R2: EMMON Project, R3: TCS Requirements Document). The Precision, Recall and F-Score for each expert as the benchmark against PROMIRAR using each requirements documents were calculated as shown in Table 5.3.

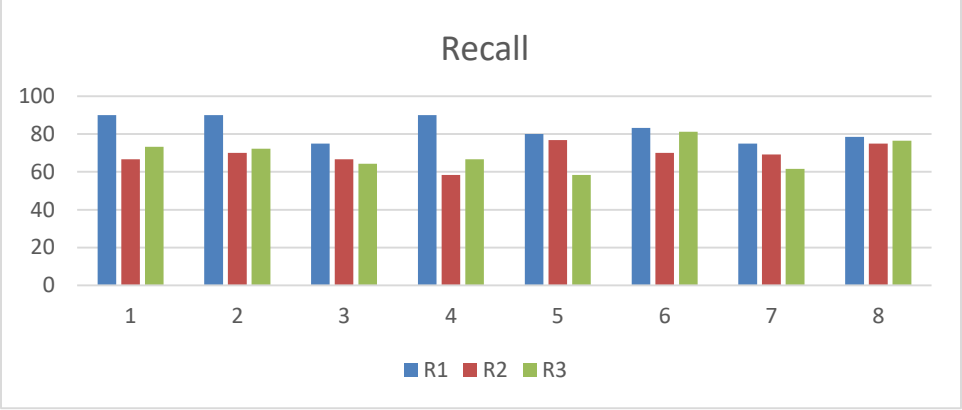
Table 5.3: Recall, Precision & F-Score Result from Experts (E1-E8) using RS1-RS3

	Requirements	E1	E2	E3	E4	E5	E6	E7	E8	Average
Precision	RS1	85	85	85	89.23	86.67	85.33	89	91.67	87.11
	RS2	86.67	88.33	83.33	88.33	83.33	86.33	85.7	85	85.88
	RS3	88.75	81.25	86.25	85	83.75	86.67	91	81.25	85.49
Average										**86.16**
Recall	RS1	90	90	85	90	80	83.33	79	78.57	84.49
	RS2	86.67	80	78.67	88.33	78.92	80	79.23	85	82.1
	RS3	83.33	82.22	84.29	86.67	88.33	81.25	81.54	76.47	83.01
Average										**83.2**
F-Score	RS1	89	87	85	88	80	83.33	80	78.57	83.86
	RS2	87.57	83	78.57	88.33	79.92	82	89.73	86	84.39
	RS3	85.36	84.22	84.59	86.47	87.36	81.25	86.54	86.47	85.28
Average										**84.51**

A chart was plotted to show the Precision and Recall result of the eight expert's outcome for each requirements documents evaluation as shown in Figure 5.4.



(a)



(b)

Figure 5.4: Precision (a) Recall (b) Chart from 8 Experts (E1-E8) using the R1-R3

5.2.6 Discussion of Performance Evaluation Results

From the evaluation results obtained using the three requirements documents, the mean precision, recall and F-score were computed with results 86.16%, 83.20% and 84.51% respectively. For a detection tool, the recall value is definitely more important than precision. In the ideal case, the recall should be 100%, as it would allow relieving human analysts from the clerical part of document analysis (Kiyavitskaya *et al*., 2008). PROMIRAR with a mean recall value of 83.20% shows that the tool is fit for practical use, as it marked a minimum of six out of eight IMR detected by a human expert and this is consistent with best practices. The mean precision of 86.16% shows that the percentage of IMR judged by experts that were also retrieved by the PROMIRAR is well above average and is consistent with best practices. The F-score, which is the harmonic mean of Precision and Recall is 84.51%. This clearly shows that PROMIRAR is very efficient. As for the IMR marked by human evaluators but missed by PROMIRAR, the manual examination has shown that they represent implicit factors where PROMIRAR could not identify explicit patterns that would allow for automated IMR detection. A further observation from the simulation experiment (see Table 5.3), revealed that the performance of the tool also depends significantly on the quality of the domain ontology (i.e. the richness of vocabulary and coverage of the ontology with respect to a specific domain increases the accuracy of PROMIRAR).

5.2.7 Comparative Evaluation of PROMIRAR and Other Tools

This section gives a comparative evaluation of PROMIRAR with other related tools. Table 5.4 shows the comparative analysis of some related tools based on the following: approach used by the tool, technologies used to perform tagging and parsing, pre-processing of requirements supported or not, and concern IMR aspect.

The tools were selected because they shared a similar purpose with PROMIRAR in that they are also used for managing an aspect of IMR (ambiguity). These tools have been used and reported in the literature and have shown significant performance evaluation in comparison to other tools to be used as a benchmark for comparison with PROMIRAR.

Table 5.4: Overview of other Relevant Tools

Tool	Approach	Technologies Used	Pre-processing	Concerned IMR aspect
NAI (Yang et al., 2010; Yang et al., 2011)	Machine learning/heuristics based	LogitBoost, Named entity recognition	Yes	Noun and Verb compound coordination, Lexical and Structural ambiguity
SR-Elicitor (Umber et al., 2011)	Controlled Language	SBVR, POS tagger	No	Lexical, Syntactic, Scope-Quantifier
ARUgue (Shah and Jinwala, 2015)	Knowledge based to ontology	WordNet, WSD	Yes	Anaphora, Coordination and Vagueness

A performance evaluation of PROMIRAR was conducted alongside these tools and the result is as shown in Table 5.5.

Table 5.5: Comparing PROMIRAR's Performance with other Selected Tools

Tools	IMR Aspect Addressed	Recall (%)		Precision (%)		F-Score (%)	
NAI	Lexical Ambiguity	70	74.2	85.4	82.36	77.73	78.28
	Structural Ambiguity	82.4	85.75	84.2	80.91	82.7	83.34
SR-Elicitor	Lexical Ambiguity	80.12	78.22	85.76	83.1	79.4	78.23
ARUgen	Vagueness	87.51	89.63	91.12	93.51	89.28	90.71

5.2.8 Discussion of Performance Evaluation Results

The tools were tested using the requirements documents in Section 5.2.1 with a focus on the concerned IMR aspects. From the comparative analysis done (see Table 5.5), it was observed that the Lexical Ambiguity and Structural Ambiguity analysis of PROMIRAR performed better than NAI in terms of Recall and F-Score but was almost at par in terms

of Precision. When the Lexical Analysis of PROMIRAR was compared with SR-Elicitor, they both performed at par across all metrics. Finally, when the vagueness analysis of PROMIRAR was compared with ARUgen it was observed that PROMIRAR performed better across all metrics.

5.2.9 Discussion of Validity Threats of Performance Evaluation

A short discussion on the validity of the performance evaluation using the expert evaluation and comparative tool evaluation is based on the categories defined by Wohlin *et al.* (2000). The threats are considered first before giving a summary of the validity of the results.

Conclusion Validity: In order to ensure reliable treatment, all participants were provided with an introduction and instructions for the experiment prior to the experiment. Also, standard measures (recall and precision) were used to assess recommendations by PROMIRAR in order to avoid misunderstanding or misinterpretation of the results. Ordinarily using eight participants in the experiment will translate to low statistical power, but for a highly technical domain like Requirements Engineering and a preliminary evaluation, this is considered to be sufficient for a first trial.

Internal Validity: A key requirement is that participants have sufficient experience or knowledge of the domain. The participants had minimum master-level education in the area of Requirements Engineering. The experts were also provided with detailed instructions on what should be done. Therefore, there were no factors other than the treatment that influenced the outcome of the experiment.

Construct Validity: In order to ensure a realistic experiment, all participants had the same instruction for the experiment. Also, they performed exactly the same task, which is to identify implicit requirements. Hence, the results obtained from participants depend only on this task (one single variable), which eliminates any mono-method bias effect.

External Validity: The key issue here is whether the results can be generalised from the preliminary evaluation to the Requirements Engineering industry. For the simulation experiment, six expert researchers all affiliated with MSU, while the industrial assessment

was done by two RE experts at software industry based in the USA. A concern could be that possibly there would have been different results if the evaluations had been performed with a bigger group of participants with more diverse background, not only in terms of coming from different institutions and countries but also with more different educational backgrounds and covering a wider spectrum of software engineering domains than could be achieved with only eight persons. The involved persons mainly had experience in RE and it is impossible to know if the tool would have been found equally promising by experts from other domains. The mitigation to this threat is to try to avoid including any domain-specific limitations in the general approach, but this does not entirely remove the threat. So, while there is currently no reason why the approach should not also be usable in other companies and other domains, an interesting point for further research is to have a wider group of experts to try out PROMIRAR.

Hence, no serious threats to validity can be foreseen for the conclusions on the evaluation performed. Also, the feedback from industry experts proved that PROMIRAR has sufficient merit for application in an industrial setting.

5.3 PROCESS FRAMEWORK EVALUATION OF PROMIRAR IN AN ORGANISATIONAL CONTEXT

It is essential to evaluate PROMIRAR in an organisational setting in order to assess its suitability to support the proposed process framework. In order to do this, an industrial case study approach was selected by using two test cases.

The structure of this section is as follows: first, the background for the case study and an overview of the requirements management tool used by the industry is introduced. This is to assess how well PROMIRAR can integrate with existing RM tools in the cause of an RE endeavour. Then, the process evaluation scenario of the case study is presented. After that, experiences of the use of the tool and improvement proposals for the improving the process is discussed.

5.3.1 Background for the Case Study

In conducting the industrial case study evaluation, two case companies were used for the process framework evaluation. The purpose of the case study was to evaluate and see how well the process for managing IMR with PROMIRAR integrates well into the requirements management framework of these companies. The evaluation also looked to find out weaknesses and problems, not just successes that might arise from the integration of PROMIRAR.

I. Overview of Case Companies

All of the companies involved in the case study were selected based on existing relationships with the researchers. Two companies were selected for the study both in the software engineering domain. The names of the companies are omitted for privacy concerns. This company will be referred to as Company A and B in this thesis.

a) Company A

Company A which is located in the US has an international standing of being a front-runner in the data centre software solutions field. It ranks amongst one of the best enterprises in Data Center software industry, with software products being used worldwide. Company A has over 18 software engineers with 5 majoring in requirements engineering. Their experience ranges from 10-15years as a RE experts. The dominant requirements management tool used is the IBM Rational RequisitePro.

b) Company B

Company B is also a US based software subsidiary that offers products for the development of engineering processes and management via Scrum. Company B develops and publishes Scrum applications lifecycle management tool. Company B also runs a ScrumCORE training section that offers Scrum training via organised community courses as well as private on-site training. Company B has well over 12 software engineers with 3 as core RE experts with experience ranging from 15-20 years. The core requirements management tool used in Company B is the CaliberRM.

II. Overview of the Requirements Management Tools

a) Tool A (IBM Rational RequisitePro)

IBM Rational RequisitePro is a requirements management tool for finding, documenting, organising, and tracking requirements. It also includes some features such as Microsoft Word and requirements database. Software project teams can gather, enter and manage requirements within the document or in a database. Some of its features enable customers and development team to establish and maintain the agreement. RequisitePro is designed for a multiuser environment.

b) Tool B (CaliberRM)

CaliberRM is a requirements management system that enables project teams to deliver higher quality applications that meet end-user specifications. It helps applications to meet end-user needs by allowing all project stakeholders (marketing teams, analysts, developers, testers, and managers) to collaborate and communicate the voice of the customer throughout the software delivery lifecycle. According to The Borland Software Requirements Definition and Management, the CaliberRM solution is an iterative and collaborative approach for defining and managing software requirements across the five critical requirements process areas—elicitation, analysis, specification, validation and management. It enables teams to fully define, manage and communicate changing requirements. Changes to requirement data are recorded and stored in the database, providing reliable and up-to-date information for effective requirements based application development and testing.

5.3.2 Approach Used to Integrate PROMIRAR with RM Tools

a) Tool A (Requisitepro)

The direct method for collecting case study was used in this research, where the researcher is in direct contact with the interviewees and collects report in real-time (Salim *et al*., 2002). The case study was done in Company A in their Requirements Engineering Lab. At the

time of the evaluation, the company was working on a project of a “virtual camera prototype” in collaboration with another company. The project included the following files:

i. The requirements specification document.
ii. The design and code files.

There are three possible approaches to the connection between PROMIRAR and RequisitePro. The first one is using RequisitePro API a shown in Figure 5.5, the second is using a direct link to the requirements database through JDBC-driver as shown in Figure 5.6 and the third approach is using RequisitePro baseline files as a source of requirements information.

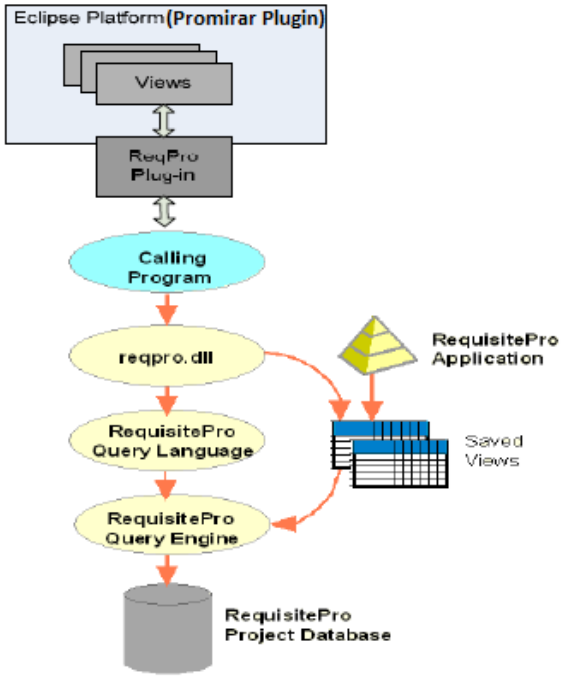


Figure 5.5: Conceptual Architecture of the First Alternative

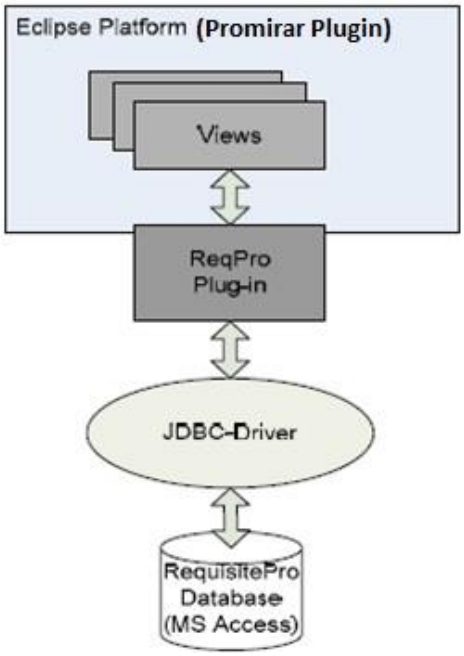


Figure 5.6: Conceptual Architecture of the Second Alternative

In going about the process for integrating PROMIRAR with Requisite Pro, the second alternative was chosen as the preferred option, since the input into PROMIRAR was the SRS document as captured in the database of Requisite Pro.

The approach works well as PROMIRAR was able to integrate with RequisitePro and access the database for retrieval of relevant SRS files.

b) Tool B (CaliberRM)

Company B was working on a *microprocessor monitoring application* when the project evaluation was carried out. In going about the integration of PROMIRAR with CaliberRM, the approach was seamless as PROMIRAR integrated with CaliberRM by directly importing the database files.

5.3.3 Method used to conduct the Case Study

A Use Case method (Heiskanen *et al*., 2006) was used to describe the separate events that took place during the case study because it helps to introduce the various operations that were done at that moment. The Use Cases described in Tables 5.6-5.9 shows the various

activities in the requirements management process while using PROMIRAR with the RM tools (RequisitePro and caliberRM).

Table 5.6: Use Case Narrative to Create a Project

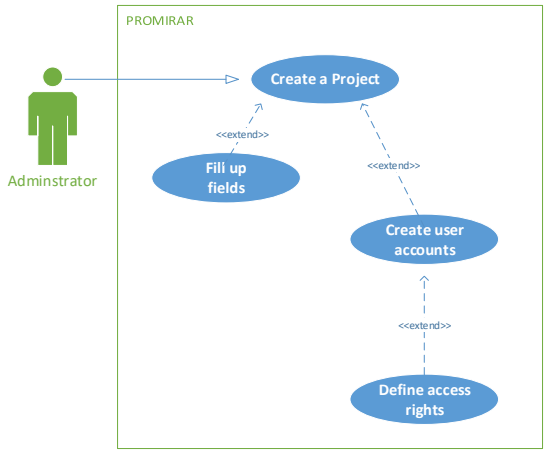
Use Case 1	**Create a Project**
Summary:	The administrator creates a project and defines user accounts and access rights for all project members.
Frequent:	Once when the project begins. Users can be added later if new project members join in the project.
Purpose:	Create a project, where requirements management issues are handled during the project. In addition, user accounts and access rights for them are defined, so that different users have appropriate access rights to the required data.
Preconditions:	PROMIRAR is configured with RM tools and it works well.
Description:	The administrator creates a project to the requirements database in a central repository. Identification fields, such as project name, database path, etc. are defined, and thereby every project can be distinguished from each other. When a project is created, the administrator creates user accounts and access rights for every user. This makes it certain that the user can edit the change requirements only if it is necessary. Access rights can be, for example, role-based or project-based.
Figure 5.7: Use Case for Creating a Project	

Table 5.7: Use Case Narrative to Import a Document

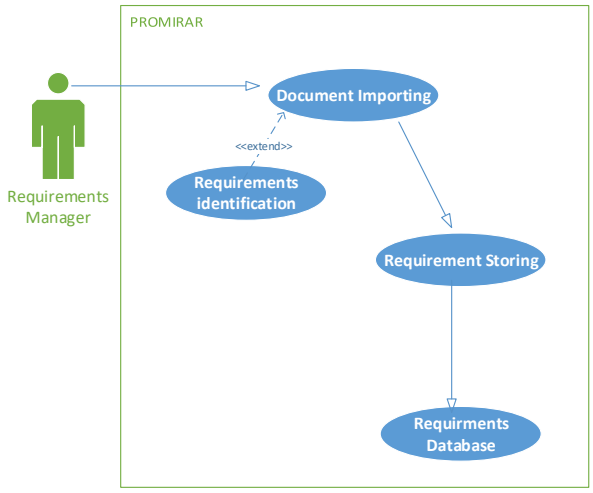
Use Case 2	**Document importing**
Summary:	Existing requirements specification document is imported to the PROMIRAR.
Frequent:	Once in the beginning of the project.
Purpose:	To import an existing requirements specification document.
Preconditions:	Requirements specification exists and project is running.
Description:	Requirements Manager imports the existing requirements specification document into PROMIRAR by using its document importing feature. In order for PROMIRAR to be able to recognise requirements from the document, they have to be identified by using certain identification methods. In practice, every requirement must be chosen by marking them manually in the document, or they can be identified by using a certain identification tag that is repeated in every requirement (e.g. REQ1, REQ2,… REQ can be the identification tag). The RM tool automatically adds the identified requirements to the database.
Figure 5.8: Use Case for importing a Document	

Table 5.8: Use Case Narrative for Requirements Management

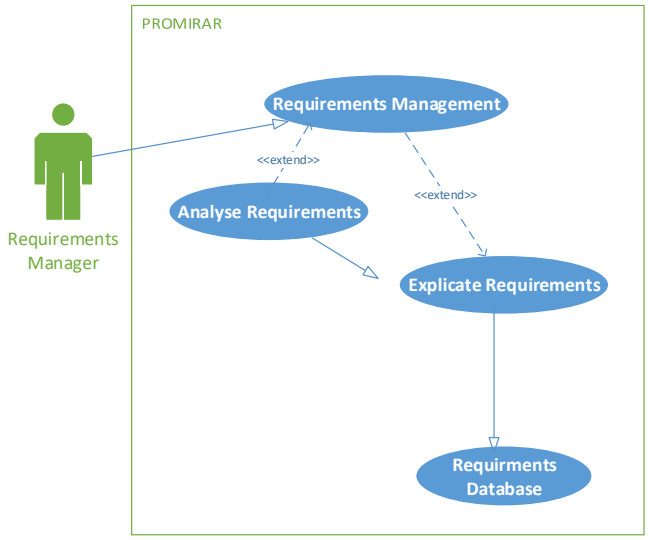
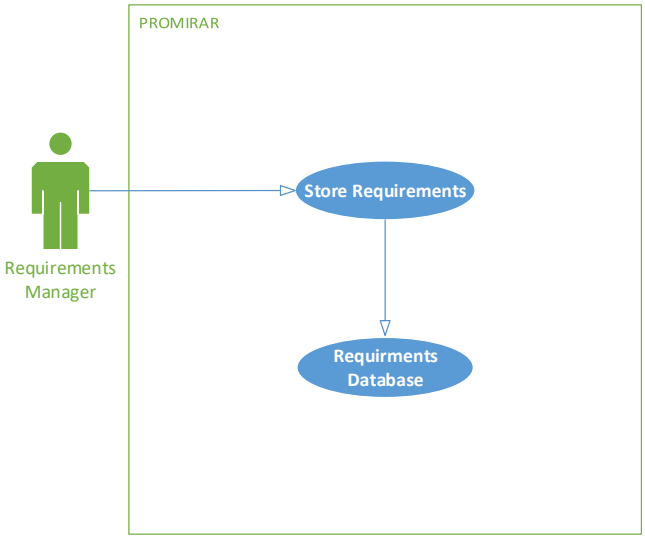
Use Case 3	**Requirements management**
Summary:	The Requirements are analysed as either explicit or implicit in nature, which means all its links and relations to the other requirements are clarified.
Frequent:	Always when a requirement is created, later if necessary.
Purpose:	To analyse all requirements, thereby making it easier to explicate all implicit requirement.
Preconditions:	The requirements, that are being managed, exist.
Description:	The requirements manager analyses all requirements. The various recommendations are revised and approved for onward use.
Figure 5.9: Use Case for Requirements Management	

Table 5.9: Use Case Narrative to add Explicated Requirements

Use Case 4	**Add explicated Requirements.**
Summary:	After analysis are concluded the explicated requirements are moved to the database.
Frequent:	Once in a while, when necessary.
Purpose:	Requirements management has been concluded.
Preconditions:	The requirements manager is satisfied with the outcome.
Description:	The explicated requirements are good enough for onward use in the software development process.
Figure 5.10: Use Case to add explicated Requirements	

5.3.4 Evaluation Report for the case studies

I. Report of Case A

During the case studies, the requirements for the Virtual Camera prototype was fed into PROMIRAR. The Virtual Camera prototype requirements documents contained 1,836 requirements both functional and non-functional. Since the project was new and no ontology existed for use. The requirements documents were used to semi-create a seed ontology for the projects. This took quite some time to create due to the large size of the requirements document. The seed ontology could not be populated due to unavailability of

domain expert given the short timeframe for the evaluation. The project team evaluator comprised of 2 experts. The requirements were analysed for IMR management. 11.2% of the requirements were found to be implicit in nature. On further explication of the discovered IMR, the explication process took a while for the RequisitePro platform before the output for the requirements was generated. This was attributed to the numerous requirements documents that PROMIRAR had to explore using its ABR module. After the explicated requirements were generated, saving it to the database on RequisitePro was problematic. This was a weakness in the RequisitePro version used. One weakness of the RequisitePro version used was its inability to add requirements in the Word document after their addition to the tool interface. Updating the document can only be done manually by cutting and pasting the addition. This makes RequisitePro unsuitable for projects where numerous requirements can be added directly through the tool interface. On the other hand, RequisitePro maintains a relationship between the repository and the requirements document when the existing requirements are updated in any way.

II. Report of Case B

The requirement of a ride sharing app was imported into PROMIRAR tool, which was integrated into CaliberRM environment. There was also no available domain ontology to use in the evaluation process. The requirements document contained 632 requirements, which took less time in comparison to RequisitePro to process in order to produce a seed ontology. The requirements were analysed for detecting IMR, 5.3% of the requirements were found to be implicit in nature. The identified implicit aspects were further explicated by the requirements engineer manually because the implicit concerns identified were less in number. The IMR and its explicated part were stored in the case base of PROMIRAR for subsequent use by other similar projects. Conducting the IMR process had a good turnaround time when compared to that of RequisitePro as CaliberRM's platform provided a feature that enabled for easy export of the requirements to its database.

III. The Procedure of the Evaluation Process and the Result

a) Description of Procedure

The industrial evaluators were asked to complete the form as shown in Table 5.11 as they make use of the PROMIRAR tool.

The evaluation process in each case was based on a four level criteria framework (González-Prida *et al*., 2011)

i. Level 1: The software does not meet the minimum criteria required.
ii. Level 2: The software does meet the minimum criteria required.
iii. Level 3: The software has largely met the minimum criteria although there are some significant weaknesses.
iv. Level 4: The software meets all the criteria required with no significant weakness.

Table 5.10 shows the evaluation report form used by each evaluator. The form gives the evaluation criteria against the various levels for which each evaluator conducted the process evaluation.

Table 5.10: Evaluation Report Form used by each Evaluator

No	Criteria	Level 1	Level 2	Level 3	Level 4
1	Import/Export of data.				
2	Integration with system and databases connection.				
3	Security. Access management and profiles				
4	Support at every stage of the analysis.				
5	Outputs of the tool				

b) Evaluation Process Results

Two (2) industrial experts completed the forms in Company A and Company B, in Table 5.11 shows the combined evaluation process result for the process evaluation conducted in

Company A and Company B using experts 1 and 2 (i.e. A1 signify expert 1 of company A, and A2 expert 2 of Company A; same rule applies to Company B with the use of B1 and B2)).

Table 5.11: Process Framework Evaluation Result by two Expert in Company A & B

No	Criteria	Level 1				Level 2				Level 3				Level 4			
		A1	A2	B1	B2	A1	A2	B1	B2	A1	A2	B1	B2	A1	A2	B1	B2
1	Import/Export of data.											√	√	√	√		
2	Integration with system and databases connection.													√	√	√	√
3	Security. Access management and profiles									√	√	√	√				
4	Support at every stage of the analysis.								√	√		√	√				
5	Outputs of the tool								√	√		√	√				

IV. Discussion of Two Case Scenarios with PROMIRAR

The evaluation of the process framework was successful as the integration of PROMIRAR with the requirement management tools and performing the IMR identification and management process produced acceptable outcomes. From the feedback form, PROMIRAR was able to import requirement documents, connect with the various databases for retrieval and saving of explicated requirements document. The various analysis produced results that showed the areas of IMR concerns.

However, the following were recommendations from the experts in order to enhance the robustness PROMIRAR:

i. Enabling other file formats such as .xls, .xml for import as PROMIRAR only provided support for importation of txt/pdf/.doc files
ii. Improving on the module for ontology creation as PROMIRAR only provided semi-automated functionality for creating seed ontology.
iii. Addressing other ambiguities such as coordination, pragmatic.
iv. Providing other output formats to support XML processing as PROMIRAR only supports .txt/pdf.

5.3.5 Discussion of Validity Threats of Industrial Case Evaluation

The results obtained in the industrial case evaluation study are presented within the strengths and limitations of the selected research methodology. Some specific validity threats are explained in this section.

Conclusion Validity

This refers to whether right conclusions can be drawn about the relationship in the data and the result obtained from the evaluation. Some of the concerns addressed in this aspect of validity are:

Construct validity: this refers to the extent to which the operational measures that are studied truly represent the theoretical constructs on which those operational measures were based (Wohlin *et al*., 2012). To achieve this, all respondents had the same instructions as a guide for completing the evaluation form. The task was the same for all, which is to complete the IMR identification form. Hence, the results obtained from the industrial evaluation depend only on one variable, which eliminates any mono-method bias effect.

Internal Validity: this refers to whether other factors other than the treatment influenced the outcome of the evaluation. For the evaluation, all respondents were software practitioners who claimed to have ample experience in requirements engineering. The two participants were fully employed software developers, who have experience in SE ranging from a minimum of 10 years.

External Validity: The key interest of this aspect of validity is whether we can generalise the outcome of the evaluation to a larger context. The respondents were mostly experienced software engineers, who have practical experience on issues that deal with implicit requirements, and have worked in top industries that do advanced software.

CHAPTER SIX

CONCLUSION AND RECOMMENDATIONS

This Chapter summarises and discusses the contribution of the thesis, and presents an outlook of the opportunities for future research work. The thesis presents a process framework for managing implicit requirements using analogy-based reasoning.

6.1 CONCLUSION

The presence of IMR in software requirements specification document has been found to pose major defects in the software development process. It has grossly affected architectural designs as well as project cost overrun.

This thesis aim of providing a process framework for managing implicit requirements within an organisation has been achieved by addressing our earlier stated objectives in Chapter one.

The first objective of empirically investigating the impact of IMR on success or otherwise of software development in practice was achieved by conducting a survey investigating the perception and handling of implicit requirements in small and medium-sized software organisations. A report on the findings from the survey was given in Chapter three. As a contribution, this thesis presents a pioneering effort that is aimed at providing an understanding of implicit requirements management practices in small and medium-sized organisations based on empirical investigation. The survey results revealed that organisational experience in terms of age in business, experience in RE, and experience of personnel in RE, and software team size have a positive correlation with effective management of implicit requirements within an organisation. The report also revealed that although the use of experience has played a significant role so far, the need for tool support is also desirable for better handling of implicit requirements. However, a significant number of practitioners believe that existing RE tools are equally sufficient for managing implicit requirements if they are maximised, and there is no need for new tools. It can be

deduced from the report that there is need to promote general understanding of implicit requirements and the need for more significant interest in issues of implicit requirements compared to explicit requirements, which have received the most attention in the literature.

The thesis further achieved the second objective of designing a process framework that both discovers and manages IMR in a systematic way as reported in Chapter three. The proposed process framework integrated three core technologies NLP, ABR and ontology for discovery and management of IMR. The framework consists of two core modules, the first module was responsible for the identification of implicit requirements in a requirements specification document while the second module was responsible for managing and explicating the identified IMR. The components of the first module comprised of the NL Processor, Feature Extractor, Ontology, and Heuristic Classifier while the component of the second module was made up of solely the analogy based reasoner. The design of this framework informed the next objective.

The third objective of providing a prototype tool support for the process framework for managing IMR was reported in Chapter Four. The tool was developed using the Eclipse IDE. Some of its associated components such as the NLP Processor made use of the Apache OpenNLP API, the ontology component made use of the Protégé-OWL API, the heuristic classifier component made use of the WordNet Java API, Gannu API and IText PDF API.

The developed tool (PROMIRAR) can be integrated into the RE process of software development organisations. This is a direct response to problems in the practice of many organisations, which have not been addressed by existing requirements management tools.

Finally, the last objective of evaluating the process framework and prototype tool using industrial case studies and controlled experiments was reported in Chapter five.

Two kinds of evaluation were performed. Firstly, an evaluation of the performance of the tool was benchmarked against both industry experts and other existing tools using three (3) different requirements specification documents in three different experiments. Secondly, an evaluation of the application of the tool as a support for the process of handling implicit requirements within a software organisation was conducted using two (2) software

development organisations as a case study. The Evaluation result showed that the tool works well with the test conducted using requirements specification from three different domains. The results also showed that the PROMIRAR performed well above average in managing IMR when compared with other tools. For the evaluation done in the industry, the tool fits well into their requirement management process. The results of the evaluation revealed that PROMIRAR produced a good outcome with respect to IMR. However, recommendations were given to help improve the tool outcome in the future.

In conclusion, the ability to discover unknown and un-elicited requirements will mitigate many risks that can adversely affect system architecture design and project cost.

This research work has reported findings from a survey of implicit requirements management practices in small and medium-sized organisations. The research also tackled two concerns in the requirement engineering domain which is the need for software developers to move from focusing on explicit requirements to seeking for practical ways of handling implicit requirements and to improve on existing requirements management tools to tackle the issue of implicit requirements.

The research has also provided a theoretical and product-oriented framework that can be leveraged for the management of implicit requirements during software developments processes.

Finally, the results of this research endeavour if adopted will give the quality boost needed in promoting the efficiency of the Requirements Engineering processes in software development organisations, by reducing software budget overrun. It will further enhance the quality of the software product delivered thereby bringing about greater user satisfaction in delivered software products.

6.2 CONTRIBUTIONS TO KNOWLEDGE

This study contributes to the general research areas of Requirements Engineering both at the global and local context. More specifically, the main contribution of this study caters for the observed limitations in existing Requirements Management approaches and tools as follows:

i. Observations and general opinions of RE practitioners suggest that IMR poses a lot of challenges for software developers, however, there is yet a lack of empirically proven evidence through research studies that have assessed the impact of IMR on the success or failure of software development projects. This research effort ranks among the first set of pioneering efforts geared at providing empirically-based evidence on the impact of implicit requirements on software process outcomes in terms of success or otherwise.

ii. Although several researchers have developed various approaches, systems or tools, which are aimed at solving the problem of requirements management, these approaches and tools lack specific provisions for managing implicit requirements (IMR). Explicit Requirements have received the most attention because of their clearly defined nature (Daramola *et al*., 2012; Kotonya and Sommerville, 1998). So far IMR is handled by the requirements engineers who use their initiative and experience to address the challenges that they pose (Jha, 2009; Parameswaran, 2011). Hence, this study offers a new approach to solving the challenges of implicit requirements in software development by evolving a systematic tool support framework which can be integrated into an organisation's Requirements Engineering procedure for managing IMR in systems development process.

iii. This research work adds to the existing body of knowledge in the area of requirements management as there is currently few relevant literature that deals with the issues of managing implicit requirements.

6.3 RECOMMENDATIONS AND FUTURE WORK

This thesis provides several opportunities for further research in the immediate future. The process framework for managing implicit requirements as implemented in this thesis found a number of issues that provides ample research possibilities to enhance the concept in the following areas:

i Empirical Survey

Firstly, there is need to evaluate the capability of existing RE management tools for managing implicit requirements, and the potentials of existing automated tools so far proposed in the literature to support management of implicit requirements throughout the RE lifecycle.

Secondly, dealing with a subject matter addressing implicitness, there is need to use mixed methods such as interviewing to complement the use of a questionnaire or using a semi-structured questionnaire rather than a closed-ended format. This would give ample opportunity for respondents to express their viewpoints (particularly in a situation where respondents know more than they can tell).

ii Enhanced Domain Ontology

This thesis made provision for the automatic creation of seed ontology from requirements specification document for a domain that does not have an ontology. There is a need for research efforts on how this seed ontology can be automatically updated so as to make for quick use instead of being enriched by the domain ontology engineer.

Furthermore, there is a need for the development of an upper level ontology of reusable software artefacts in several domains.

iii Ambiguity Detection Component

There is need to increase the number of ambiguities covered as the current tool only addresses lexical, syntactical/structural ambiguity. Other forms of ambiguities such as Pragmatic ambiguity which occurs when a sentence has several meanings in the context in

which it is uttered; Semantic ambiguity which occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity. This will enrich the IMR identification module of the PROMIRAR tool.

REFERENCES

Aamodt, A., and Plaza, E. (1994): Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, **7(1):** 39-59.

Abel B., Maimon. O (2015): Ontology Learning from Text. *International Journal of Signs and Semiotic Systems* **4:** 21-14.

Abma, B. J. M. (2009): Evaluation of requirements management tools with support for traceability-based change impact analysis. Master's thesis, University of Twente, Enschede.

Ahamed, S. R. (2010): An Integrated and comprehensive approach to software quality. *International Journal of Engineering Science and Technology*, **1(2):** 59-66.

Al-Harbi, O., Jusoh, S., and Norwawi, N. (2012): Handling ambiguity problems of natural language interface for question answering. *International Journal of Computer Science Issues (IJCSI)*, **9(3):** 245-265.

Antoniou, G., and Van Harmelen, F. (2004): Web ontology language: Owl. In *Handbook on ontologies* (pp. 67-92). Springer Berlin Heidelberg.

Aranda, J., Easterbrook, S., and Wilson, G. (2007): Requirements in the wild: How small companies do it. *In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International* (pp. 39-48). IEEE.

Arnold, R.S. and Bohner, S.A. Eds. (1996): Software Change Impact Analysis. Los Alamitos, California, USA, *IEEE Computer Society Press*.

Bajwa, I. S., Lee, M., and Bordbar, B. (2012): Resolving syntactic ambiguities in natural language specification of constraints. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 178-187). Springer Berlin Heidelberg.

Bartha, P. (2010): By parallel reasoning. Oxford University Press.

Barzdins J, Barzdins G, Cerans K, Liepins R, Sprogis A. (2010): OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2. *OWLED 2010 Proceedings of 7th International Workshop OWL; Experience and Directions.* (pp. 614).

Berry, D. M., and Kamsties, E. (2004): Ambiguity in requirements specification. In *Perspectives on software requirements* (pp. 7-44). Springer US.

Berry, D. M., Kamsties, E., and Krieger, M. M. (2003): From contract drafting to software specification: Linguistic sources of ambiguity. *A Handbook.*

Banerjee, S., and Pedersen, T. (2002): An Adapted Lesk algorithm for word sense disambiguation using WordNet. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 136-145). Springer Berlin Heidelberg.

Bao, D. (2008): A Survey on Similarity-based Reasoning. http://www4.ncsu.edu/~dbao/ Analogy Survey_final.pdf

Bontas, E. P., Mochol, M., Tolksdorf, R. (2005): Case studies on ontology reuse. In *Proceedings of the IKNOW05 International Conference on Knowledge Management* (pp. 345–353). Graz, Austria.

Brandt, S. C., Morbach, J., Miatidis, M., Theißen, M., Jarke, M., and Marquardt, W. (2008): An ontology-based approach to knowledge management in design processes. *Computers & Chemical Engineering*, **32(1):** 320-342.

Bussel, D. V. (2009): Detecting ambiguity in requirements specifications. (Doctoral dissertation, Master's thesis, Tilburg University).

Castañeda, V., Ballejos, L., Caliusco, M. L., and Galli, M. R. (2010): The use of ontologies in requirements engineering. *Global Journal of Research in Engineering*, **10(6):** 2-8.

Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999): What Are Ontologies? Why Do We Need Them? *IEEE Intelligent Systems*, **14(1):** 20-26.

Choi, F. Y. (2000): Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference* (pp. 26-33). Association for Computational Linguistics.

Chomsky, N. (1956): Three models for the description of language. *IRE Transactions on information theory*, **2(3):** 113-124.

Chopra, A., Prashar, A., Sain, C. (2013): Natural language processing. *International Journal of Technology Enhancements and Emerging Engineering Research*, **1(4):** 131-134.

Cimiano, P. (2006): Ontology Learning and Population from Text. Algorithms, Evaluation and Applications, ISBN: 978-0-387-30632-2, *Springer*, New York.

Copi, I. and Cohen C. (2005): Introduction to Logic. 12th edition, Upper Saddle River, New Jersey: Prentice-Hall.

Daramola, O., Moser, T., Sindre, G., and Biffl, S. (2012): Managing implicit requirements using semantic Case-based Reasoning research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 172-178). Springer Berlin Heidelberg.

Davies, T., and Russell, S. J. (1987): A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan, Morgan Kaufmann Publishers, Inc (pp. 264-270).

De Mantaras, R. L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., and Keane, M. (2005): Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, **20(03):** 215-240.

Deshpande, S., and Richardson, I. (2009): Management at the outsourcing destination-global software development in India. *In Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on* (pp. 217-225). IEEE.

Dobson, G., and Sawyer, P. (2006): Revisiting ontology-based requirements engineering in the age of the semantic web. In *Proceedings of the International Seminar on Dependable Requirements Engineering of Computerised Systems at NPPs* (pp. 27-29).

Douglass, D.: Understanding Implicit Requirements of Software Architecture (06-08-2009), http://alturl.com/wauae

Dreyer, H., Wynn, M. G., and Bown, G. R. (2015): Tacit and Explicit Knowledge in Software Development Projects: Towards a Conceptual Framework for Analysis. In *eKnow 2015 7th International Conference on Information, Process and Knowledge Management* (pp. 49-52). ThinkMind.

Drysdale, D. (2007): High-Quality Software Engineering: Lessons from Six-Nines World. Lulu.com.

Duncker, K. (1945): On problem-solving (L. S. Lees, Trans.).*Psychological Monographs*, **58(5):** 270.

Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. (2001): An automatic quality evaluation for natural language requirements. In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ* **1:** 4-5.

Falco, R., Gangemi, A., Peroni, S., Vitali, F. (2014): Modelling OWL ontologies with Graffoo. In Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (Eds.), The Semantic Web: ESWC 2014 Satellite Events, *Lecture Notes in Computer Science* **8798:** 320–325. Berlin, Germany: Springer. DOI: 10.1007/978-3-319-11955-7_42

Gacitua, R, Ma, L., Nuseibeh, B., Piwek, P., de Roeck, A.N., Rouncefield, M., Sawyer, P., Willis, A and Yang, H, (2009): Making tacit requirements explicit. In *Managing Requirements Knowledge (MARK), 2009 Second International Workshop on* (pp. 40-44). IEEE.

Garcia and N. Medinilla (2007): The ambiguity criterion in software design. In *International Workshop on Living with Uncertainties (IWLU'07)*. ACM.

Genesereth, M. R., and Nilsson, N. J. (1987). Logical foundations of artificial. *Intelligence. Morgan Kaufmann*, 58.

Gentner, D. (1983): Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, **7(2):** 155-170

Ghezzi, C., and Nuseibeh, B. (1998): Special Section-Managing Inconsistency in Software Development-Guest Editorial: Introduction to the Special Section. *IEEE Transactions on Software Engineering*, **24(11):** 906-907.

Gleich, B., Creighton, O., and Kof, L. (2010): Ambiguity Detection: Towards a tool explaining ambiguity sources. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 218-232). Springer Berlin Heidelberg.

Gliem, R. R., and Gliem, J. A. (2003): Calculating, interpreting, and reporting Cronbach's alpha reliability coefficient for Likert-type scales. *Midwest Research-to-Practice Conference in Adult, Continuing, and Community Education* (pp. 82-88). Ohio, USA.

Glinz, M., and Fricker, S. A. (2015): On shared understanding in software engineering: an essay. *Computer Science-Research and Development*, **30(3-4):** 363-376.

Goguen, J. and Jirotka, M. (Ed.) (1994): Requirements Engineering: Social and Technical Issues, London: Academic Press.

Gomez-Perez, A., Fernández-López, M., and Corcho, O. (2006): Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. *Springer Science & Business Media*.

González-Prida, V., Carlos, C., and Barberá, A. C. L. (2011): Review and Evaluation Criteria for Software Tools Supporting the Implementation of the RCM Methodology. *International Journal of E-Business Development.* (pp. 18–27).

Gorschek, T., and Svahnberg, M. (2005): Requirements Experience in Practice: Studies of Six Companies. *In Engineering and Managing Software Requirements*. (pp. 405-426) Springer Berlin Heidelberg.

Gotel, O. C., and Finkelstein, C. W. (1994): An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on* (pp. 94-101). IEEE.

Grehag, Å.(2001): Requirements Management in a Life-Cycle Perspective - A Position Paper. In: *Proceedings of the 7th International Workshop on REFSQ* 2001, Interlaken, Switzerland, (pp. 183–188).

Gruber, T. (2009): Ontology in Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds.), *Springer-Verlag*, 2009.

Gruninger M. and Lee J. (2002): Ontology: Applications and Design. *Communications of the ACM*, **45(2):** 39.

Guarino, N. (1998): Formal ontology and information systems. In *Proceedings of FOIS* (Vol. **98**, No. 1998, pp. 81-97).

Hadad, G., Doorn, J., Kaplan, G. (2009): Explicitar Requisitos del Software usando Escenarios. In: *Proceedings WER"09, Workshop on Requirements Engineering*.

Heiskanen, N., Raatikainen, K., and Heinonen, S. (2006): Fetal macrosomia–a continuing obstetric challenge. *Neonatology*, **90(2):** 98-103.

Hesse, M. B. (1966): Models and Analogies in Science. Notre Dame: University of Notre Dame Press

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004): Design Science In Information Systems Research. *MIS Quarterly*, **28 (1):** 75-105.

Holyoak, K. J. (1985): The pragmatics of analogical transfer. *Psychology of Learning and Motivation*, **19:** 59-87. New York: Academic Press.

Holyoak, K. J., and Thagard, P. (1989): Analogical mapping by constraint satisfaction. *Cognitive Science*, **13(3):** 295-355.

Ihme, T., Pikkarainen, M., Teppola, S., Kääriäinen, J., and Biot, O. (2014): Challenges and industry practices for managing software variability in small and medium sized enterprises. *Empirical Software Engineering*, **19(4):** 1144-1168.

Jantunen, S. (2010): Exploring software engineering practices in small and medium-sized organisations. *In Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering ACM* (pp. 96-101).

Jayadianti, H., Pinto, C. S., Nugroho, L. E., Santosa, P. I., and Widayat, W. (2013): Solving Different Languages Problem (Portuguese, English, and Bahasa Indonesia) in Digital Library with Ontology. In: *International Conference on Information & Communication Technology and Systems (ICTS)*, Bali.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, **117(2):** 277-296.

Kauppinen, M., Vartiainen, M., Kontio, J., Kujala, S., and Sulonen, R. (2004): Implementing requirements engineering processes throughout organisations: success factors and challenges. *Information and Software Technology*, **46(14):** 937-953.

Leffingwell, D., and Widrig, D. (2000): Managing software requirements: a unified approach. Addison-Wesley Professional.

Lehmann, J., and Völker, J. (Eds.) (2014): Perspectives on Ontology Learning, Studies on the Semantic Web. AKA Heidelberg /IOS Press.

Ljunglof, P., and Wirén, M. (2010): Syntactic parsing. In *Handbook of Natural Language Processing, Second Edition* (pp. 59-91). Chapman and Hall/CRC.

Lung, C. H., Urban, J. E., and Mackulak, G. T. (2007): Analogy-based domain analysis approach to software reuse. *Requirements Engineering*, **12(1):** 1-22.

Jha, R.: Gathering Implicit Requirements (10-06-2009), http://alturl.com/ocyb5

Jurisica, I., Mylopoulos, J., and Yu, E. (2004): Ontologies for knowledge management: an information systems perspective. *Knowledge and Information Systems*, **6(4):** 380-401.

Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., & Hendler, J. (2006): Swoop: A web ontology editing browser. Web Semantics: Science, *Services and Agents on the World Wide Web*, **4(2):** 144-153.

Kamsties, E., Berry, D. M., and Paech, B. (2001): Detecting ambiguities in requirements documents using inspections. In *Proceedings of the first workshop on inspection in software Engineering (WISE'01)* (pp. 68-80).

Kamsties, E., Hörmann, K., and Schlich, M. (1998): Requirements engineering in small and medium enterprises. *Requirements engineering*, **3(2):** 84-90.

Kano, N., Nobuhiku, S., Fumio, T., Shinichi, T.(1984): Attractive Quality and Must-be Quality. *Journal of the Japanese Society for Quality Control* **14(2):** 39–48.

Kapoor, B., and Sharma, S. (2010): A comparative study ontology building tools for semantic web applications. *International Journal of Web & Semantic Technology (IJWesT)*, **1(3):** 1-13.

Karatas, E., K., (2012): An Ontology-Based Approach to Requirements Reuse Problem in Software Product Lines. Master's Thesis, *Middle East Technical University*.

Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., and Hendler, J. (2006): Swoop: A web ontology editing browser. *Web Semantics: Science, Services and Agents on the World Wide Web*, **4(2):** 144-153.

Keynes, J.M. (1921): A Treatise on Probability, London: Macmillan.

Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. M. (2008): Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, **13(3):** 207-239.

Korner, S. J., and Brumm, T. (2009): Resi-a natural language specification improver. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on* (pp. 1-8). IEEE.

Kossmann, M., and Odeh, M. (2010): Ontology-driven requirements engineering a case study of ontorem in the aerospace context. INCOSE. In *International Symposium*.

Kotonya G. and Sommerville, I. (1998): Requirements Engineering: Processes and Techniques. Chichester, UK: *John Wiley & Sons*.

Lami, G., Gnesi, S., Fabbrini, F., Fusani, M., and Trentanni, G. (2004): An automatic tool for the analysis of natural language requirements. *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*.

Land, L. P. W., Aurum, A., and Handzic, M. (2001): Capturing implicit software engineering knowledge, in *Software Engineering Conference Proceedings*, Australian, pp. 108-114. IEEE.

Lang, M., and Duggan, J. (2001): A tool to support collaborative software requirements management. *Requirements Engineering*, **6(3):** 161-172. Springer-Verlag London Limited.

Larsson, A., Steen, O. (2008): Tool Support for Requirements Management Quality from a User Perspective. In: *Proceedings of IRIS29*, Helsingör, Denmark

Lehmann, J., and Voelker, J. (2014): An introduction to ontology learning. Perspectives on ontology learning. AKA/IOS Press, Heidelberg, 9-16.

Liddy E.D., (2001): Natural Language Processing. *2 ed. Encyclopaedia of Library and Information Science*. NY. Marcel Decker, Inc.

Lin, C., and Wu, C. (2005): Managing knowledge contributed by ISO 9001: 2000. *International Journal of Quality & Reliability Management*, **22(9):** 968-985.

Lin, J., Fox, M. S., and Bilgic, T. (1996). A requirement ontology for engineering design. *Concurrent Engineering*, **4(3):** 279-291.

Ljunglöf, P. and Wirén, M. (2010): Syntactic parsing. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing,2nd edition, chapter 4. CRC Press, Taylor and Francis*. ISBN 978-1420085921

Lopez, O., Laguna, M.A., Garcia, F.J. (2002): Metamodeling for requirements reuse. *WER02—Workshop em Engenharia de Requisitos*. Valencia, Spain, (pp. 76–90).

Loucopoulos, P., and Karakostas, V. (1995): System requirements engineering. McGraw-Hill, Inc.

Lubars, M., Potts, C., and Richter, C. (1993): A Review of the State of the Practice in Requirements Modeling. *In Requirements Engineering, Proceedings of IEEE International Symposium on* (pp. 2-14). IEEE.

Maedche, A., and Staab, S. (2002). Measuring similarity between ontologies. In *International Conference on Knowledge Engineering and Knowledge Management* (pp. 251-263). Springer Berlin Heidelberg.

Maedche, A.D., (2003): Ontology Learning for the Semantic Web, Norwell, Massachusetts, Kluwer Academic Publishers.

Maiden, N. A., and Rugg, G. (1996): ACRE: selecting methods for requirements acquisition. *Software Engineering Journal*, **11(3):** 183-192.

Maiden, N. (1998): CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, **5(4):** 419-446.

Maiden, N. A. M. and Sutcliffe, A. G. (1992): Exploiting Reusable Specifications through Analogy. *Communications of the ACM*, **34(5):** 55-64.

Maiden, N. (1991): Analogy as a Paradigm for Specification Reuse. *Software Engineering Journal* **6(1):** 3–15.

Mala, G. A., and Uma, G. V. (2006): Object oriented visualization of natural language requirement specification and NFR preference elicitation. *IJCSNS*, **6(8):** 91.

Malone, J., Brown, A., Lister, A. L., Ison, J., Hull, D., Parkinson, H., and Stevens, R. (2014): The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of biomedical semantics*, **5(1):** 25.

Maniraj, V., and Sivakumar, R. (2010): Ontology languages-A review. *International Journal of Computer Theory and Engineering*, **2(6):** 887.

Marrero, M., Sánchez-Cuadrado, S., Fraga, A., and Llorens, J. (2008): Applying Ontologies and intelligent text processing in requirements reuse. In *First Workshop on knowledge reuse (KREUSE'08)* (pp. 25-29).

Matulevičius, R. (2005): Survey of requirements engineering practice in Lithuanian software development companies. *Information Systems Development*, 327-339.

Meyer, B. (1985): On formalism in specifications. *IEEE Software*, **2(1):** 6.

Mitzias, P., Riga, M., Kontopoulos, E., Stavropoulos, T. G., Andreadis, S., Meditskos, G., and Kompatsiaris, I. (2016): User-Driven Ontology Population from Linked Data Sources. In *International Conference on Knowledge Engineering and the Semantic Web* (pp. 31-41). Springer International Publishing.

Mizoguchi R, Sunagawa E, Kozaki K, Kitamura Y. (1997): A Model of Roles within an Ontology Development Tool. Hozo. *Journal of Applied Ontology*. **2(2):** 159-79.

Mizoguchi, F. and Ohwada, H., (1995): Using Constraint Programming to Design an Option-based Decision Support System. *Intelligent Systems in Accounting, Finance and Management*, **4:** 13–26.

Mohammed, A. (2008): Concept relation extraction using natural language processing: the CRISP technique. *Theses and Dissertations*. Iowa State University.

Monzon, A. (2008): A Practical Approach to Requirements Reuse in Product Families of On-Board Systems. In: *International Requirements Engineering*, pp. 223–228, IEEE Press.

Mooney, R. J. (2014): CS 343: Artificial Intelligence Natural Language Processing, University of Texas at Austin. Available at: http://www.cs.utexas.edu/ mooney/cs343/slide-handouts/nlp.pdf.

Moore, B. N., and Parker, R. (1997): Critical Thinking Instructor's Manual: The Logical Accessory. Mayfield Publishing Company.

Naval surface warfare center, Dahlgren Division - Naval Sea Systems (12-07-2000) www.navsea.navy.mil/Portals/103/Documents/NSWC_Dahlgren/.../E3/E3.pdf

Nikula, U., Sajaniemi, J., and Kälviäinen, H. (2000): A State-of-the-practice Survey on Requirements Engineering in Small and Medium-sized Enterprises. Lappeenranta, Finland: Lappeenranta University of Technology.

Noman I, Mohammed SS, Zubair AS. (2010): TODE. A Dot Net Based Tool for Ontology Development and Editing. ICCET 2010. *Proceedings of the 2nd International Conference on Computer Engineering and Technology*; Chengdu: China. p. 229-3.

Nonaka, I., and Von Krogh, G. (2009): Perspective—Tacit knowledge and knowledge conversion: Controversy and advancement in organisational knowledge creation theory. *Organisation Science*, **20(3):** 635-652.

Nuseibeh, B., and Russo, A. (1999): Using abduction to evolve inconsistent requirements specification. *Australasian Journal of Information Systems*, **6(2)**.

Nuseibeh, B., and Easterbrook, S. (2000): Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46). ACM.

Odeh, M. (2009): Towards a universal requirements engineering process. In: *The International Arab Conference on Information Technology*, Yemen.

Onyeka, E. (2013): A process framework for managing implicit requirements using analogy-based reasoning: Doctoral consortium paper. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)* (pp. 1-5). IEEE.

Palmer, D. D. (2010): Text preprocessing. In *Handbook of Natural Language Processing*, Second Edition (pp. 9-30). Chapman and Hall/CRC.

Parameswaran, A.: Capturing Implicit Requirements (02-08-2011), http://alturl.com/emeej

Pittke, F., Leopold, H., and Mendling, J. (2015): Automatic detection and resolution of lexical ambiguity in process models. *IEEE Transactions on Software Engineering*, **41(6):** 526-544.

Poesio, M (2000): Semantic Analysis. In R. Dale, H. Moisl and H. Somers (eds.), *Handbook of Natural Language Processing*. Marcel Dekker.

Pohl, K. (1997): Requirement Engineering, In *A. Kent, J. Williams, C.M. Halls (eds), Encyclopaedia of Computer Science and Technology*, M. Dekker, New York, vol **36:** 345-386.

Polyani, M. (1983): The Tacit Dimension. Paul Smith Publishing ISBN 0-8446-5999-1.

Popescu, D., Rugaber, S., Medvidovic, N., and Berry, D. M. (2008): Improving the quality of requirements specifications via automatically created object-oriented models. *Innovations for Requirements Engineering*, **71**.

Popescu, D., Rugaber, S., Medvidovic, N., and Berry, D. M. (2008): Reducing ambiguities in requirements specifications via automatically created object-oriented models.

In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs* (pp. 103-124). Springer Berlin Heidelberg.

Popper, K. (2009): Science: Conjectures and refutations. *The philosophy of science: an historical anthology*, 471.

Quispe, A., Marques, M., Silvestre, L., Ochoa, S. F., and Robbes, R. (2010): Requirements engineering practices in very small software enterprises: A diagnostic study. *In Chilean Computer Science Society (SCCC), 2010 XXIX International Conference of the IEEE* (pp. 81-87).

Rech, J., Ras, E., and Decker, B. (2007): Intelligent assistance in German software development: A survey. *IEEE Software*, **24(4):** 72-79.

Renkema, J. (2004): Introduction to Discourse Studies. Amsterdam/Philadelphia: John Benjamin Publishing Company.

Salim, M. D., Villavicencio, A., and Timmerman, M. A. (2002). A method for evaluating expert system shells for classroom instruction. *Journal of Industrial Technology*, **19(1):** 1-11.

Sambasiva R., (1992): "A Model based Approach to Analogical Reasoning and Learning in Design" Thesis Proposal for Georgia Institute of Technology.

Sanderson, M. (2010): Test collection based evaluation of information retrieval systems. *Foundations and Trends® in Information Retrieval*, **4(4):** 247-375.

Schneider, R. J. (2002): System and software requirements validation through inspections: constructive reading and mining requirements from natural language requirements documents. *Information Knowledge Systems Management*, **3(2-4):** 173-194.

Shah, U. S., and Jinwala, D. C. (2015): Resolving ambiguity in natural language specification to generate UML diagrams for requirements specification. *International Journal of Software Engineering, Technology and Applications*, **1(2-4):** 308-334.

Shah, U. S., and Jinwala, D. C. (2015): Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, **40(5):** 1-7.

Shaw, M. L. and Gaines, B. R. (1996): Requirements acquisition. *Software Engineering Journal*, **11(3):** 149-165.

Siegmund, N., Rosenmuller, M., Kastner, C., Giarrusso, P. G., Apel, S., and Kolesnikov, S. S. (2011): Scalable prediction of non-functional properties in software product lines. In *Software Product Line Conference (SPLC), 2011 15th International* (pp. 160-169). IEEE.

Singer, L., Brill, O., Meyer, S., Schneider, K. (2009): Utilising Rule Deviations in IT Ecosystems for Implicit Requirements Elicitation. In: *Proceedings of the Second International Workshop on Managing Requirements Knowledge (MaRK)*, pp. 22–26.

Singh, S., and Saikia, L. P. (2015): Ambiguity in Requirement Engineering Documents: Importance, Approaches to Measure and Detect, Challenges and Future Scope. *International Journal of Advanced Research in Computer Science and Software Engineering*, **5(10)**, ISSN: 2277 128X

Sinha, S., and Husain M. S.(2016): Proposal for Avoiding Ambiguity in Requirement Engineering using Artificial Intelligence. In *Proceedings of the ACEIT Conference*. pp 1-4.

Software Requirements Specification – EMMON (12-07-2010) http://www.artemis-emmon.eu/deliverables/FP7-JU-EMMON-2010-DL-WP7-003-D7.1-software-requirements-specification-document.pdf.

Solemon, B., Sahibuddin, S., and Ghani, A. A. A. (2010): Adoption of requirements engineering practices in Malaysian software development companies. *In International Conference on Advanced Software Engineering and Its Applications* (pp. 141-150). Springer, Berlin, Heidelberg.

Spanoudakis, G. (1996): Analogical reuse of requirements specifications: A computational model. *Applied Artificial Intelligence*, **10(4):** 281-305.

Spanoudakis, G., and Zisman, A. (2001): Inconsistency management in software engineering: Survey and open research issues. In *Handbook of Software Engineering and Knowledge Engineering, World Scientific*, 329–380.

Stoiber, R., & Glinz, M. (2009): Modelling and managing tacit product line requirements knowledge. In *Managing Requirements Knowledge (MARK), 2009 Second International Workshop on*, pp. 60-64. IEEE.

Stojkovic, M., Trifunovic, M., Misic, D., and Manic, M. (2015): Towards analogy-based reasoning in semantic network. *Computer Science and Information Systems*, **12(3):** 979-1008.

Stone, A., and Sawyer, P. (2006): Identifying tacit knowledge-based requirements. *IEE Proceedings-Software*, **153(6):** 211-218.

Sure, Y., Angele, J., and Staab, S. (2003): OntoEdit: Multifaceted inferencing for ontology engineering. In *Journal on Data Semantics I* (pp. 128-152). Springer Berlin Heidelberg.

Sureephong, P., Chakpitak, N., Ouzrout, Y., and Bouras, A. (2008): An ontology-based knowledge management system for industry clusters. In *Global Design to Gain a Competitive Edge* (pp. 333-342). Springer London.

Swoop. Semantic Web Ontology Overview and Perusal. 2004. Available from: http://www.mindswap.org/2004/SWOOP/

TCS Software Requirements Specification (02-12-1999) https://fas.org/irp/program/collect/docs/sss_20.pdf.

Tjong, S. F (2008): Avoiding ambiguity in requirements specifications.Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy.

Tjong, S.F., Hallam., N. and Hartley, M. (2006): Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns. In *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT06)*, pp. 199.

Thörn, C. (2010): Current state and potential of variability management practices in software-intensive SMEs: Results from a regional industrial survey. *Information and Software Technology*, **52(4):** 411-421.

Umber, A., Bajwa, I. S., and Naeem, M. A. (2011): NL-based automated software requirements elicitation and specification. In *International Conference on Advances in Computing and Communications* (pp. 30-39). Springer Berlin Heidelberg.

W3C.TopBraid. (2001) Retrieved from W3C: http://www.w3.org/2001/sw/wiki/TopBraid

Welcome to Apache OpenNLP (24/10/2012) http://opennlp.apache.org/

Wei, C. H., Harris, B. R., Kao, H. Y., and Lu, Z. (2013): tmVar: a text mining approach for extracting sequence variants in biomedical literature. *Bioinformatics*, **29(11):** 1433-1439.

Wilson, W. M., Rosenberg, L. H., and Hyatt, L. E. (1997): Automated analysis of requirement specifications. In *Proceedings of the 19th International conference on Software engineering* (pp. 161-171). ACM.

Winkler, S., and Pilgrim, J. (2010): A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)*, **9(4):** 529-565.

Woods, J., Irvine A., and Walton D. (2004): Argument: Critical Thinking, Logic and the Fallacies, 2nd edition, Toronto: Prentice-Hall.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012): Experimentation in software engineering. *Springer Science & Business Media*.

Xu, Q.L., Jiao, R.J., Yang, X., Helander, M.G., Khalid, H.M., Anders, O. (2007): Customer Requirement Analysis Based on an Analytical Kano Model. In: *Industrial Engineering and Engineering Management*, pp. 1287–1291. IEEE Press.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., and Nuseibeh, B. (2011): Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering*, **16(3):** 163.

Yang, H., Willis, A., De Roeck, A., and Nuseibeh, B. (2010): Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 53-62). ACM.

Yatskevich, M., and Giunchiglia, F. (2004): Element level semantic matching using WordNet. In *Meaning Coordination and Negotiation Workshop, ISWC*

Youn, S., and McLeod, D. (2006): Ontology development tools for ontology-based knowledge management. *In Encyclopedia of E-Commerce, E-Government, and Mobile Commerce* (pp. 858-864). IGI Global.

Yu, L., Kittur, A., and Kraut, R. E. (2014): Distributed analogical idea generation: inventing with crowds. *In Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (pp. 1245-1254). ACM.

Zave, P. (1997): Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, **29(4):** 315-321.

Zave, P., and Jackson, M. (1997): Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **6(1):** 1-30.

Zhdanova A.V and Keller, U (2005): Choosing an ontology language. *In Proceedings of World Academy of Science, Engineering and Technology*. **2: 47-50**.

APPENDIX A

Requirements Documents used for Evaluation

Requirements for Course Management System (CMS)

The CMS used to determine the performance of the tool described in Section 1 contains the following sixteen requirements:

Req1: The system shall allow end-users to provide profile and context information for registration.

Req2: The system shall provide functionality to search for other people registered in the system.

Req3: The system shall provide functionality to allow end-users to log in the system with their password.

Req4: The system shall support three types of end-users (administrator, lecturer and student).

Req5: The system shall allow lecturers to set an alert on an event.

Req6: The system shall maintain a list of events the students can be notified about.

Req7: The system shall notify the students about the occurrence of an event as soon as the event occurs.

Req8: The system shall actively monitor all events.

Req9: The system shall notify students about the events of the lectures they are enrolled for.

Req10: The system shall allow students to enrol for lecturers.

Req11: The system shall allow lecturers to send e-mail to students enrolled for the lecture given by that lecturer.

Req12: The system shall allow assigning students to teams for each lecture.

Req13: The system shall allow lecturers to send e-mail to students in the same group.

Req14: The system shall allow lecturers to modify the content of the lectures.

Req15: The system shall give different access rights to different types of end-users.

Req16: The system shall support two types of end-users (lecturer and student) and it shall provide functionality to allow end-users to log in the system with their password.

Requirements for Tactical Control System (TCS) System/Subsystem Specification

Brief Description: The TCS consists of the software, software-related hardware and the extra ground support hardware necessary for the control of the Outrider, and the Predator UAV, and future tactical UAVs. The TCS provides hardware and software necessary to allow operators conduct the following major functions 1) mission planning, 2) mission control and monitoring, 3) payload product management, 4) targeting, and 5) C4I system interface.

1. The TCS shall have the functionality to allow the operator to generate a UAV mission plan.
2. The TCS shall have the functionality to receive and process UAV mission plans from service specific mission planning systems.
3. The TCS Mission plan shall include all necessary information required to be interoperable with the service specific mission planning systems including the Tactical Aircraft Mission Planning System (TAMPS), Aviation Mission Planning System (AMPS), and Air Force Mission Support System (AFMSS).
4. The TCS shall have the functionality to transmit UAV mission plans to service specific mission planning systems.
5. The TCS shall facilitate automated processing of mission plan data received via C4I interfaces in order to extract the appropriate mission planning data.

6. The TCS shall have the functionality to receive and process UAV mission plans from other TCSs.
7. The TCS shall have the functionality to transmit UAV mission plans to other TCSs.
8. The TCS shall be capable of storing a minimum of 500 mission plans under unique names to allow for later retrieval.
9. The TCS mission planning function shall provide a graphical user interface that gives the operator the ability to define waypoints on a map based display using a pointing device with full keyset redundancy.
10. The TCS shall provide the capability to compute the range and bearing between two geographic positions on the map display.
11. The TCS shall permit dynamic mission and payload retasking during all phases of operational mission execution.
12. The TCS shall allow the operator to enter as well as review mission plan parameters, including AV flight parameters, payload control parameters, data link control parameters, AV VCR control parameters (if applicable to the selected AV), and AV loiter patterns.
13. The TCS shall provide the capability to enter system configuration characteristics in the mission plan, to include selected AV type, AV identification number, selected payload type, ground control authorization information, and required communications pre-set for data links, tactical communications, and C4I data dissemination.
14. The TCS shall provide the system functionality necessary to upload a flight route plan and payload plan (if applicable) to the AV via the selected system data link as well as direct ground connection. [SSS070]
15. TCS shall provide the capability for the operator to retrieve a mission plan for viewing, modification, as well as deletion at the operator's discretion [SSS071], and allow the operator to save the mission plan under a different name, for future retrieval [SSS072].
16. The TCS shall automatically check the validity of the intended mission plan prior to being uploaded including altitude constraints, payload constraints, data link

range constraints, airspace restrictions, fuel limitations, threat constraints, data link terrain masking effects, and Loss of Link (LOL) Plan. [SSS073]

17. The TCS shall notify the operator of all discrepancies found during the mission plan check as well as indicate successful completion of the mission plan check. [SSS074]
18. The TCS shall provide the capability to override validation faults after the fault is acknowledged by the operator. [SSS540]
19. The TCS shall allow the operator to set the LOL delay timer(s) during mission planning. [SSS075] The LOL delay is the time from when the AV detects an unplanned LOL to the time it initiates LOL procedures.
20. The TCS shall provide the capability to print waypoint data in alphanumeric format.

Requirements for Smart City (Embedded Monitoring)

Brief Description: The EMMON project is a European Research and Development (R&D) project. EMMON motivation is originated from the increasing societal interest and vision for smart locations and ambient intelligent environments (smart cities, smart homes, smart public spaces, smart forests, smart parking system etc.). The development of embedded technology allows for smart environments creation and scalable digital services that increase the human quality of life.

1. The C&C shall provide the users with real--time data regarding the measured values, as collected from the various sensors part of the network.
2. The C&C shall support the configuration of ranges for sensor readings (maximum and minimum allowed values).
3. The C&C shall report potential sensor malfunctions to the users when the reading is "Suspicious" or "Invalid".
4. The C&C shall allow users to validate readings that were qualified as "Suspicious" or "Invalid". This means that users shall be allowed to qualify as "Good", sensor readings that were classified as "Suspicious" or "Invalid" automatically.
5. The C&C shall notify users if there are manually modified values, whenever it presents sensor data to them.

6. The C&C shall have the sensor readings displayed in a GIS environment.
7. The C&C shall represent the sensor nodes in the system as two--dimensional sets of dots (or symbols) in a rectangular panel.
8. The C&C shall provide a visual display of sensor readings to the users, by clicking on each sensor node's representation.
9. The C&C shall allow for the visual selection of elements of interest by using layers of information.
10. Each layer shall be associated with a particular type of element of interest.
11. The C&C shall set the appropriate endangerment level, according to the sensor readings.
12. The C&C shall provide the users with historical data regarding the measured values.
13. The C&C shall keep a history of collected sensor readings of up to 1 year.
14. The C&C shall allow the management (create, update, delete) of user accounts.
15. The C&C shall allow the definition of different access privileges for each account type.
16. The C&C shall allow loading a file with spatial data (shapefile) containing nodes.
17. The C&C shall allow users to setup the sensors' operating parameter.
18. The C&C shall allow for scheduling of node maintenance procedures.
19. The C&C shall allow distribution of data to the relevant authorities, through SMS and email.
20. The C&C shall provide a geographical visualisation of all areas in alarm/alert status.
21. The C&C shall provide the values of parameters at most 30 seconds after the reading was ordered by the user and the data has been received from the Sensor Network after it has been requested by the C&C.
22. The C&C shall update the information available to the users every minute.
23. The C&C shall access the interface provided by the EMMON middleware for retrieving sensor data.
24. The C&C shall access the interface provided by the event propagation module.
25. The C&C shall support multi--language. The default language shall be English.

APPENDIX B

The Questionnaire Form

The purpose of this survey is to assess the impact of Implicit Requirements (IMR) on the success or failure of requirements engineering during software development. IMR is the hidden or assumed requirements that usually do not get mentioned by stakeholders during requirements elicitation but which a system is expected to fulfil, in order to be wholly accepted by users. Some opinions seem to suggest that IMR throws up substantial challenges during software development, this survey seeks to empirically investigate the impact of IMR on software development.

Section 1: Introductory Questions

Company

1. Name of Company/Organisation ……
2. Name of Country……
3. Number of years of business
   - ○ 0 – 5 years
   - ○ 6 – 10 years
   - ○ 11 – 15 years
   - ○ 16 – 20 years
   - ○ above 20 years
4. Size of software development team
   - ○ 0 – 5 years
   - ○ 6 – 10 years
   - ○ 11 – 15 years
   - ○ 16 – 20 years
   - ○ 21 – 50 years
   - ○ above 50 years

5. The domain of software development?

- ☐ Business/enterprise information systems
- ☐ System software/operating systems
- ☐ Game engines/computer games/graphics software
- ☐ Computer packages
- ☐ Web/Internet-based development
- ☐ Mobile apps and software
- ☐ Industrial automation software
- ☐ Control software and embedded systems
- ☐ Telecommunications software
- ☐ Military and defence software systems
- ☐ Middleware/ CASE tool
- ☐ others……

6. The target market of the software product?

- ☐ Local/Domestic only
- ☐ International/global
- ☐ Both Local and International.

Organisation's Background

7. Your professional status in your organisation

- ☐ Junior level
- ☐ Middle level
- ☐ Managerial level

8. Your experience in Requirements Engineering (RE) practice
   - ○ 0 – 5 years
   - ○ 6 – 10 years
   - ○ 11 – 15 years
   - ○ 16 – 20 years
   - ○ above 20 years
9. Experience of your organisation in RE
   - ○ 0 – 5 years
   - ○ 6 – 10 years
   - ○ 11 – 15 years
   - ○ 16 – 20 years
   - ○ above 20 years
10. Specific area(s) of expertise in RE

- ☐ Requirements elicitation
- ☐ Requirements analysis
- ☐ Requirements modeling
- ☐ Requirements specification
- ☐ Requirement validation
- ☐ Requirements management

Section 2: Process Requirements Engineering

Please answer the following question as appropriate and applicable.

2.1. IMR, if not well managed can have a negative impact on the quality of software product?

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.2. Lack of proper management of IMR can lead to cost/budget overrun during software product?

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.3. IMR does not have any effect on the acceptability of software product?

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree

☐ Strongly Disagree

2.4. IMR does not have any impact on the correctness of system architecture?

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.5. Relying principally on experience is sufficient to the discovery of IMR requirements during requirements elicitation

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.6. Using experience plus tool support will be perfect for managing IMR

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.7. A specialised approach, possibly with some automation support will be useful for managing IMR

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.8. Improper handling of IMR can lead to poor system design and poor product performance

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.9. Proper handling of IMR will reduce maintainability of software products

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.10. My organisation uses specialised approach for handling IMR

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.11. The approach for handling IMR in my organisation is sufficient

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.12. My organisation use a specific requirements prioritisation technique(s) to identify IMR

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.13. There is no need to evolve new methods to specially handle IMR

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.14. Established RE management methods are adequate to handle IMR for now

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.15. During requirements elicitation, stakeholders deliberately withhold certain information which creates IMR scenarios

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

2.16. Stakeholders inadvertently fail to mention IMR during requirements elicitation

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

2.17. Tacit knowledge which stakeholders find difficult to express is often a cause of IMR

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

APPENDIX C

List of Publications from the Thesis

1. Emebo, O., Daramola, O., & Ayo, C. (2017): A survey on implicit requirements management practices in small and medium-sized enterprises. Tehnički vjesnik, 24(Supplement 1), 219-227.
2. Emebo, O., Olawande, D., & Charles, A. (2016): An automated tool support for managing implicit requirements using Analogy-based Reasoning. In Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on (pp. 1-6). IEEE.
3. Emebo, O., Varde, A. S., & Daramola, O. (2016): Common Sense Knowledge, Ontology and Text Mining for Implicit Requirements. In Proceedings of the International Conference on Data Mining (DMIN) (p. 146). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
4. Onyeka, E. (2013): A process framework for managing implicit requirements using analogy-based reasoning: Doctoral Consortium paper. In Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on (pp. 1-5). IEEE.