Representing Requirements Traceability using XML Topic Maps

David Hollings
dholling@cs.uct.ac.za

Supervised by:

Justin Kelleher

Department of Computer Science
University of Cape Town

October 2005
Abstract

Despite the great importance of requirements traceability to the development of successful software systems, there are few systems or applications that provide useful requirements management techniques. This thesis looks at XML Topic Maps as a new technique for the representation of requirements traceability in order to improve requirements management and visualisation.

The aim of this project is to investigate the validity of using XML Topic Maps to represent requirements traceability. A software application is developed and used to prove this concept.

The system and project goals are then tested. It is shown that the use of XML Topic Maps is probably an improved method for the management and visualisation of requirements traceability. Furthermore, it is concluded that an improved visualisation technique will be required for the intuitive navigation of the requirements traceability graphs.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ IV
LIST OF TABLES .............................................................................................................. IV

CHAPTER 1: INTRODUCTION ......................................................................................... 1
  1.1 PROJECT OBJECTIVES ............................................................................................ 1
  1.1.1 Research Question ............................................................................................... 1
  1.1.2 Additional Objectives ......................................................................................... 1
  1.1.3 Academic Contribution ....................................................................................... 1
  1.2 PROJECT PLAN ........................................................................................................ 1
  1.3 PROJECT OVERVIEW ............................................................................................. 3
  1.3.1 System Overview ................................................................................................. 3
  1.3.2 Work Breakdown Structure ............................................................................... 4
  1.3.3 Divergence from the Proposal ............................................................................ 4
  1.4 REPORT OUTLINE .................................................................................................. 4

CHAPTER 2: BACKGROUND .......................................................................................... 6
  2.1 REQUIREMENTS MANAGEMENT ........................................................................... 6
  2.1.1 Introduction .......................................................................................................... 6
  2.1.2 Background ......................................................................................................... 6
  2.1.3 Description .......................................................................................................... 7
  2.2 UNIFIED MODELLING LANGUAGE ...................................................................... 8
  2.2.1 Introduction .......................................................................................................... 8
  2.2.2 Description .......................................................................................................... 8
  2.3 REQUIREMENTS TRACEABILITY ......................................................................... 10
  2.3.1 Introduction .......................................................................................................... 10
  2.3.2 Description .......................................................................................................... 11
  2.3.3 Importance of Requirements Traceability .......................................................... 13
  2.3.4 Problems with Requirements Traceability .......................................................... 14
  2.4 TOPIC MAPS .......................................................................................................... 15
  2.4.1 Introduction .......................................................................................................... 15
  2.4.2 Background .......................................................................................................... 16
  2.4.3 Description .......................................................................................................... 16
  2.4.4 Topic Maps Syntax .............................................................................................. 18
  2.4.5 Implementations ................................................................................................. 22
  2.5 BACKGROUND SUMMARY .................................................................................. 24

CHAPTER 3: SYSTEM DESIGN ...................................................................................... 25
  3.1 RATIONAL UNIFIED PROCESS ......................................................................... 25
  3.1.1 Overview ............................................................................................................. 25
  3.1.2 Application .......................................................................................................... 27
  3.2 PRECONCEPTION .................................................................................................. 29
  3.2.1 UML Modelling of Requirements ...................................................................... 29
  3.2.2 Requirements Traceability Matrix ....................................................................... 30
  3.2.3 Exporting UML Models ....................................................................................... 31
  3.3 SYSTEM REQUIREMENTS ................................................................................... 31
  3.3.1 Requirements Elicitation ...................................................................................... 31
  3.3.2 Use-case Modelling ............................................................................................. 32
  3.4 SYSTEM OVERVIEW ........................................................................................... 39
  3.4.1 High Level Overview ........................................................................................... 39
  3.4.2 Low Level Overview ........................................................................................... 40
  3.5 FILE MANAGER .................................................................................................... 41
  3.5.1 Project Creator .................................................................................................... 42
CHAPTER 4: IMPLEMENTATION

4.1 FILE MANAGER ................................................................. - 45 -
4.1.1 Project Creator ........................................................... - 45 -
4.1.2 Project Opener ............................................................ - 46 -
4.2 PARSER .................................................................. - 46 -
4.3 REQUIREMENTS MANAGER ........................................... - 47 -
4.4 TRACEABILITY MANAGER ............................................... - 48 -
4.5 INTEGRATION AND FINAL SYSTEM ............................. - 49 -
4.5.1 Integration ................................................................. - 49 -
4.5.2 Final System .............................................................. - 50 -
4.6 TRACEABILITY OF IMPLEMENTATION ............................. - 51 -

CHAPTER 5: TESTING AND VALIDATION

5.1 TESTING TECHNIQUES AND TYPES ..................................... - 53 -
5.1.1 Testing Methodology ..................................................... - 53 -
5.1.2 Functional Testing ....................................................... - 54 -
5.1.3 XTM Validation Testing ................................................. - 54 -
5.1.4 Heuristic Evaluation .................................................... - 55 -
5.1.5 User Testing ............................................................... - 55 -
5.1.6 Traceability of Test Cases ............................................ - 56 -
5.2 TEST CASES .................................................................. - 57 -
5.2.1 Functional Testing ....................................................... - 57 -
5.2.2 Validation of Outputs .................................................. - 57 -
5.2.3 Proof of Concept Testing ............................................. - 58 -

CHAPTER 6: RESULTS AND FINDINGS

6.1 RESULTS OF TESTING ......................................................... - 59 -
6.1.1 Functional Testing ....................................................... - 59 -
6.1.2 Validation of Outputs .................................................. - 59 -
6.1.3 Proof of Concept Testing ............................................. - 59 -

CHAPTER 7: CONCLUSIONS

7.1 THE HYPOTHESIS IS PROBABLY TRUE ............................ - 61 -
7.2 THE TNAV VISUALISATION OF TOPIC MAPS IS A PROBLEM - 61 -
7.3 REQUIREMENTS TRACEABILITY CAN BE EFFECTIVELY MANAGED USING XML TOPIC MAPS. - 61 -
7.4 THE FUNCTIONALITY OF THE SYSTEM IS ADEQUATE .......... - 62 -

CHAPTER 8: LESSONS LEARNT AND FUTURE WORK

8.1 LESSONS LEARNT ............................................................. - 63 -
8.1.1 Topic Maps provide an intuitive visualisation of requirements traceability ........................................ - 63 -
8.1.2 A generalised overview of all requirements and their relationships is required ................................. - 63 -
8.2 FUTURE WORK ............................................................... - 63 -
8.2.4 Future work on this thesis ............................................ - 63 -
8.2.5 Future work in related research ................................... - 64 -

CHAPTER 9: REFERENCES .............................................................. - 66 -
LIST OF FIGURES

FIGURE 1: PROJECT TIMELINE ................................................................. - 2 -
FIGURE 2: SYSTEM OVERVIEW ............................................................... - 3 -
FIGURE 3: COMMON REQUIREMENTS PROBLEMS ................................... - 7 -
FIGURE 4: CLASSES AND RELATIONSHIPS .......................................... - 9 -
FIGURE 5: A SIMPLE TOPIC MAP ......................................................... - 18 -
FIGURE 6: RUP OVERVIEW ................................................................. - 25 -
FIGURE 7: ITERATIVE DEVELOPMENT TIMELINE .................................. - 28 -
FIGURE 8: REQUIREMENTS IN UML ....................................................... - 30 -
FIGURE 9: A TRACEABILITY MATRIX .................................................... - 30 -
FIGURE 10: TRACEABILITY OF REQUIREMENTS ..................................... - 32 -
FIGURE 11: HIGH-LEVEL SYSTEM USE-CASE ....................................... - 33 -
FIGURE 12: PROJECT MANAGEMENT USE-CASE .................................... - 34 -
FIGURE 13: PROJECT CREATION USE-CASE ......................................... - 34 -
FIGURE 14: OPEN PROJECT USE-CASE .................................................. - 35 -
FIGURE 15: REQUIREMENTS TRACEABILITY USE-CASE ....................... - 36 -
FIGURE 16: MANAGE REQUIREMENTS USE-CASE .................................. - 37 -
FIGURE 17: MANAGE TRACEABILITY USE-CASE .................................... - 38 -
FIGURE 18: TRACEABILITY OF USE-CASES .......................................... - 38 -
FIGURE 19: HIGH-LEVEL SYSTEM OVERVIEW ...................................... - 39 -
FIGURE 20: LOW-LEVEL SYSTEM OVERVIEW (PACKAGES) ..................... - 40 -
FIGURE 21: FILE MANAGER PACKAGE ............................................... - 41 -
FIGURE 22: THE PARSER PACKAGE ...................................................... - 42 -
FIGURE 23: THE REQUIREMENTS MANAGER PACKAGE ......................... - 43 -
FIGURE 24: THE TRACEABILITY MANAGER PACKAGE ............................ - 44 -
FIGURE 25: FINAL PROJECT CREATOR INTERFACE ................................ - 45 -
FIGURE 26: FINAL PROJECT OPENER INTERFACE ................................... - 46 -
FIGURE 27: FINAL REQUIREMENTS MANAGER INTERFACE ................... - 48 -
FIGURE 28: FINAL TRACEABILITY MANAGER INTERFACE ...................... - 49 -
FIGURE 29: FINAL SYSTEM INTERFACE .............................................. - 50 -
FIGURE 30: TRACEABILITY OF IMPLEMENTATION ................................... - 52 -
FIGURE 31: TESTING HIERARCHY ......................................................... - 53 -
FIGURE 32: TRACEABILITY OF TEST CASES ......................................... - 56 -

LIST OF TABLES

TABLE 1: RUP DISCIPLINES AND WORKFLOWS ...................................... - 27 -
TABLE 2: FUNCTIONAL TESTING TECHNIQUES ...................................... - 54 -
TABLE 3: XTM VALIDATION TECHNIQUES .............................................. - 54 -
TABLE 4: HEURISTIC EVALUATION TECHNIQUES .................................... - 55 -
TABLE 5: USER TESTING TECHNIQUES .................................................. - 55 -
Chapter 1: Introduction

This chapter introduces a summary of this thesis. A project hypothesis is presented along with related objectives. The academic contribution is stated and a project overview is given thereafter.

1.1 Project Objectives

1.1.1 Research Question

The hypothesis of this project is whether requirements traceability modelled using UML can be successfully visualised using XML Topic Maps and whether this representation adds value to the field of requirements traceability.

1.1.2 Additional Objectives

The objectives of this project are:

- To model requirements and requirements traceability using UML
- To visually represent this UML Model using XML Topic Maps
- To manipulate and modify the model contained within the XML Topic Map

1.1.3 Academic Contribution

This project contributes to the realm of software engineering and traceability by providing research and evaluation of a potential method to improve requirements traceability visualization and navigation.

By validating the research question from section 1.1.1 this project will provide a model that could be used for the intuitive representation and impact analysis of requirements traceability in order to provide better requirements management during software development.

1.2 Project Plan

Figure 1 gives a brief overview of the project timeline. Initial tasks consisted of a preconception and concept validation phase. Thereafter, system development consisting of continued research, system development using the Rational Unified Process and continuous documentation took place. Finally, the report was completed along with the required poster.
This project is comprised of four distinct phases that occur over the above linear project timeline. Each phase is further broken up into separate sections as detailed below:

1. **Background research and analysis for the verification of the projects core concepts.**
   - Background research into requirements traceability
   - Background research into XML Topic Maps
   - Background research into tools and concepts to support the core concepts of the project

2. **Preconception of the project to derive an overview of the tasks to be performed.**
   - Requirements traceability modelling in UML
   - Manual modelling and visualisation of XML Topic Maps

3. **Development of a software tool to support the premises upon which this project is based.**
   - Using the Rational Unified Process to drive software development
   - Concurrent documentation

4. **Summary of the project and completion of the final deliverables.**
   - Completion of this document
   - Poster Design
   - Project Demonstrations
   - Completion of webpage
   - Completion of technical paper
1.3 Project Overview

This project consists of five complete sections:

- Modelling of requirements traceability in UML
- Conversion of the UML model to a XML Topic Maps syntax
- Visualisation of the XML Topic Map containing the requirements traceability model
- Management of requirements traceability using the XML Topic Map syntax
- Filtering and analysis of requirements traceability using the XML Topic Map syntax.

1.3.1 System Overview

Figure 2: System Overview

Figure 2 shows the system diagram containing the five distinct sections of the project as well as the flow of information throughout the system.
1.3.2 Work Breakdown Structure

Both group members will be covering certain visualisation aspects of the XML Topic Map syntax since it is this concept that forms the core of both projects and is required for validation of sections in each project. The five sections have been split between the two group members in the following way:

Kieran Sharpey-Schafer

Kieran’s project and research will consist of:

- Conversion of UML models to the XML Topic Map syntax
- Visualisation of the XML Topic Map containing the requirements traceability.
- Filtering and analysis of the requirements traceability using the XML Topic Maps syntax

David Hollings

David’s project and research will consist of:

- Modelling of requirements traceability using UML
- Visualisation of the XML Topic Map containing the requirements traceability
- Managing of the requirements traceability using the XML Topic Map syntax.

1.3.3 Divergence from the Proposal

Although this project closely resembles that stated in the original project proposal, there are some differences between it and the proposal.

The differences occurred due to research conducted regarding requirements traceability and visualisation thereof using XML Topic Maps.

The project follows the original specification of modelling requirements traceability using UML. Instead of conducting a case study, however, it was found more useful to conduct research into XML Topic Maps and their usefulness for visualising and managing requirements traceability.

1.4 Report Outline

This project describes the modelling of requirements traceability in UML, the subsequent requirements traceability visualisation and navigation using XML Topic Maps as well as the management and manipulation of the requirements, requirement attributes and traceability relationships contained within the topic map.
Chapter 2 introduces and provides background to the concepts used throughout this report.

Chapter 3 highlights the design of a tool used in a supporting role of the projects goals and concepts.

Chapter 4 discusses the implementation of the tool described.

Chapter 5 concerns testing to validate the developed tool along with concepts in this project.

Chapter 6 presents the results of the testing.

Chapter 7 presents the conclusions of this thesis.

Chapter 8 suggests future work on this project and the concepts involved herein.
Chapter 2: Background

This chapter introduces the reader to the various concepts at the core of this thesis. An introduction of requirements management, the Unified Modelling Language, requirements traceability and finally the concept of a Topic Map is given. The chapter aims to inform the reader with an intention to given an overview of the concepts as well as the significant relation between them. A summary is given at the end of this chapter.

2.1 Requirements Management

2.1.1 Introduction

The primary measure of a software projects success is the extent to which it fulfils its reasons for creation. This is the aim of Requirements Engineering (RE) and Requirements Management (RM). [35]

A requirement is a condition that a created project must address and satisfy [43]. Thus the systems compliance with all requirements determines the success or failure of that project [43].

Requirements engineering, as defined by Zave, “is the branch of software engineering concerned with the 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 behavior, and to their evolution over time and across software families [35].”

Requirements management, as defined by Kruchten [24], “is a systematic approach to eliciting, organizing, communicating, and managing the changing requirements of a software-intensive system or application”.

Requirements management has always been a critical aspect of any software development project [43, 47].

2.1.2 Background

During the 1990’s a new view of requirements engineering and requirements management evolved. This evolution of thinking in terms of requirements as system functions, constraints and goals saw the introduction of the IEEE sponsored conference and symposium and the establishment of an international journal published by Springer [35].

According to Nuseibeh and Easterbrook [35], this change also saw the introduction of three important new concepts:
1. The need for requirement elicitation in the organisational and social contexts in which the new system would have to operate
2. The need to describe the environment in which the system would have to operate and, thus, the specific functions and constraints required.
3. Attempts to build consistent requirement models were futile due to changing requirements and scope.

2.1.3 Description

Requirements management is critical to the success of any software development project. Meeting project requirements provides success while failure to do so increases the possibility of failure. Various reports to support this have been published in recent years [36, 50]:

- The Standish Group’s CHAOS Reports from 1994 and 1997 established that the most significant contributors to project failure relate to requirements.
- In December 1997, Computer Industry Daily reported on a Sequent Computer Systems, Inc. study of 500 IT managers in the U.S. and U.K. that found 76 percent of the respondents had experienced complete project failure during their careers. The most frequently named cause of project failure was “changing user requirements.”

Oberg et al [36] highlight four areas of concern, as shown in figure 3 below.

![Figure 3: Common Requirements Problems](image)
The graph shows the four major areas resulting in failures associated with requirements management. As can be seen, the major areas of concern for the effective management of project requirements are:

- The inability to track changes in requirements result in out-of-date specifications and incorrect implementations.
- The difficulty in eliciting stakeholder requirements and capturing them in documents to assist in the software development process results in incomplete requirement specifications.
- The existing problem of feature creep whereby new requirements are constantly appearing results in great management difficulties.
- The problems surrounding the organisation of requirements in easy to view and navigate models make for difficulties in interpreting and implementing a correct system.

Rosenberg et al. [47] indicate that problems associated with requirements are at least 14 times more costly to correct if found during testing than if they were discovered during the requirements phase. As projects near completion, the costs associated with correcting erroneous requirements increase dramatically.

Nuseibeh and Easterbrook [35] suggest that the most useful benefit of modelling requirements is the ability to analyse and monitor them. The ability to interpret and draw conclusions from visual instances of requirement models allow for possibilities to solve certain problems associated with areas of concern as highlighted by Oberg et al.

### 2.2 Unified Modelling Language

#### 2.2.1 Introduction

The Unified Modelling Language [8] (UML) is a complete set of modelling notations as specified by the Object Modelling Group (OMG) [15]. Over recent years, UML has gained a high level of acceptance in the architectural design community.

Though it was originally intended for detailed design, UML’s ability to describe entities and the relationships between them have allowed for its potential application in a wide variety of fields and research [18].

#### 2.2.2 Description

The Unified Modelling Language is described as general-purpose visual modelling language that is designed to specify, visualize, construct, and document the artifacts of a software system [32].

UML is based on a series of diagrams used for visualisation. These models include [32]:

- 8 -
These UML models are described by MacKinnon and Murphy [LU5] as representations of real world subjects. UML diagrams can further add value to technical documentation by allowing the user to organise, visualise and understand complex programming concepts.

One of the more useful UML diagrams for modelling and visualising static real world objects is the class diagram.

Classes within UML class diagrams can represent objects and the interrelations between such objects [32]. Classes are drawn as rectangles with a name, a section for attributes and a section for operations. These classes are linked together by means of dependency relationships, generalizations or associations [8].

2.2.2.1 Dependency

A dependency relationship is depicted by a dotted line with an open head showing that one entity depends on another entity. Dependencies can be used to show instantiations or dependency on inputs [8]
2.2.2.2 Generalization

Inheritance

A generalization inheritance is depicted by a solid line with an arrowhead that points from a sub-class to a superclass [8].

2.2.2.3 Association

Associations show that classes are related and are therefore aware of each other. A bi-directional association indicates that both objects in the relationship are aware of each other, while a unidirectional association indicates that only one of the objects in a relationship is aware of the other. Two types of specialised associations exist. These are aggregations and compositions [8].

Aggregation

An association aggregation is depicted by a solid line with a hollow diamond at the tail. An aggregation indicates that one object uses another object and is therefore not responsible for its creation or destruction [8].

Composition

An association composition is represented by a solid line with a solid diamond at the tail end. A composition indicated that one object owns the other and is responsible for its creation and destruction of another object [8].

2.3 Requirements Traceability

2.3.1 Introduction

Requirements traceability, as defined by Gotel and Finkelstein, is the ability to describe and follow the life of a requirement in both a forwards and backwards direction - from its
origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases [51].

Requirements traceability allows us to ensure that stakeholders requirements are correctly implemented at all stages of a software development process [28]. Throughout such a software development process, requirements are always changing. These changes and their impacts often pose as great challenges to software developers [51, 28, 31, 27].

In an iterative software development process, requirements and the traces between them change. If the implementation of a requirement has an impact on other requirements, a trace dependency should exist between them [14]. Trace dependencies describe the relationships existing between various requirements along with the associated impact they have on one another [46].

In order to keep all requirements up to date, changes within a specific requirement should result in subsequent changes in all requirements on which they impact [27].

2.3.2 Description

Gotel and Finkelstein [17] identified the traceability problem as “the inability to locate and access the sources of requirements and [pre-Requirements Specification] work”. They identified this as the major cause of other outstanding problems, i.e.:

- Out-of-date Requirements Specifications
- Slow realisation of change
- Poor reuse

As a solution, they recommended communication between parties responsible for requirements specifications and requirements management [4]. However, despite advances in many fields related to software engineering, knowledge engineering and knowledge management, many aspects of the traceability problem are still current [4].

In order to address these problems, several things are import for creating more meaningful traceability dependencies [12]:

- **Textual rationale**: In addition to a direct link between requirements, the ability to attach a textual description of the reason for the link assists in determining the relationship being described.
- **Grouping of links**: In many cases, single links between requirements cannot fully display the relationship. The ability to group links based on certain characteristics assists in describing the relationship fully.
- **Typing of link groups**: Formal types used to define links or groups of links aid in analysing traceability and traceability relationships.

There are many different types of relationships that exist between requirements, however, each individual type of relationship provides little information.
For example, tracing dependencies from individual customer requirements to design documentation and from design documentation to code modules is useful, however, without a trace dependency from requirements to implemented code modules, the entire process still requires humans to deduce the trace dependencies. In this situation, to trace the implementation of requirements in specific code modules would require tracing requirements to design documentation first and then manually verifying each requirement's implementation in code [49].

In order to avoid such problems the following rules can be used for asserting relationships between trace dependencies:

- **Bi-Directionality**: If A traces to C then C must trace to, at least, A.
- **Transitivity**: If A traces to B and B traces to C then transitively A also traces to C.
- **Commonality**: if A is known to trace to some elements CA and B is known to trace to some elements CB then a trace dependency exists if CA and CB overlap. [13]

Thus, a change in a requirement would result in subsequent changes in all requirements dependent on it and, in turn, trigger changes in all requirements dependent on them and so on.

Further, if there exists a trace dependency from A to C and a trace dependency from B to C, then there exists a trace dependency from A to B. This would occur since a change in A would trigger a change in C which would subsequently trigger a change in B. The use of common elements is thus another powerful property of trace dependencies [13, 49].

A critical issue is monitoring this change in requirements and their subsequent effects throughout the system. This concerns monitoring the validity and conformance of trace dependencies over time as requirements keep changing [31]. The process in charge of supervising such changes in requirements is known as requirements management (RM) [28].

Effective requirements management is an important issue for successful traceability in the realm of software engineering. Successful software engineering is subject to the ability to trace requirements throughout design down to specific test cases [41].

Thus, neglecting traceability or implementing insufficient traces or procedures could decrease system quality, increase development time and negatively impact on functionality [51].

Common traceability practices make use of various approaches such as matrices, hyperlinks or commercial applications [11]. Various problems to these approaches exist. These problems will be discussed in detail at a later stage.
2.3.3 Importance of Requirements Traceability

A study by the Standish Group showed that thirty-one percent of software development projects are cancelled before they are completed [26]. Many development methodologies have suggested strategies for successful and timely software development, however, many projects are late, over-budget or cancelled [26].

Traceability between aspects of a software development process can be used to combat such problems. There are many reasons why traceability can be useful, some of which are:

- It provides an aid to understanding the code.
- It provides an aid to understanding the requirements.
- It provides an aid to understanding why the code does or does not work correctly. This is particularly important in safety and mission critical applications. [46]

In essence, traceability is essential for many purposes, including conformance of a final system to its requirements and the use of standard notations and terms [2].

An efficient process for creating and maintaining trace dependencies has been identified as the hyper linkage of documents from different phases of the project. Intuitive browsing and navigation of such a repository could provide a meaningful historical guide and development aid [51]. The use of such linkage between paragraphs of documents or actual requirements is a great improvement over traditional methods of traceability navigation. Despite a general unidirectional approach, it enables analysis of change impact and propagation throughout a software project [12].

Working downwards from requirements, change impact analysis can be used to monitor change propagation throughout a system. Working upwards towards requirements can help assess if all aspects of design are covered by the requirements [12].

Traceability can provide important insight into system development and evolution. Antoniol et al. maintain that traceability assists in both top-down and bottom-up program comprehension [11]. According to Jacobson, Booch, and Rumbaugh, traceability aids both understanding and change [11]. Palmer asserts that traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation [11]. Finally, requirements traceability is a critical component in the long term maintenance of any medium or large scaled software application [11].

2.3.4 Problems with Requirements Traceability

Although traceability is agreed to be essential, it is often avoided both through unfamiliarity and because it is burdensome [2]. Traceability can be hard to achieve in
practice. Reluctance to maintain documentation, and traceability links can easily be broken if changes are not propagated throughout the system and documentation is not correctly updated [11, 46].

While traceability is tedious to construct manually, some automated requirements management tools provide only the bare minimums for creating traceability links while most provide only the ability to manually create traceability relationships with no support for the creation of new relationships from existing ones [4, 3]. Further, current tools lack support for impact analysis and change implementation. As a result, the implementation of changes throughout a system is error prone and could be very expensive [23].

To date, the manual tracing between use cases and requirements, along with the necessary training and associated limitations results in the slow production of traceability relationships [2]. Further, since traceability usually comprises of many-to-many relationships, there are typically many more traces than requirements [2, 11]. The visualisation and navigation of such a traceability hierarchy can be slow and tedious, generally resulting in the implementation of smaller, incomplete hierarchies [11].

When changes are called for and implemented, justification is required for all design decisions made. This justification is important for understanding decisions and impacts of changes throughout the system by current and future work. There are three problems identified with justification:

- **Increased Burden:** Overworked engineers do not take kindly to being told to record the reason for every decision they make
- **Accessibility:** It's not sufficient to document the design justification: this justification must be accessible when you really need to know why a decision was made
- **Freshness:** Out-of-date justification is potentially more dangerous than no justification at all: thus changes to the system will require changes to the traceability information. [4]

There are, however, even larger information management problems present in the design of most complex systems. Information models provided by CASE tools are so large and complex, that in large projects, one engineer is in charge of the tool and all communication goes through him [4]. This results in a lack of available information or a lack of updates to traceability relationships due to time constraints.

Increasingly systems are being developed concurrently by a consortium of companies. Direct communication is harder and, despite the available technology, engineers do resort to communication via formal paper-based documentation, which effectively places a barrier on traceability. Though some success has been achieved in the area of distributed traceability there are still some barriers to more widespread use [4].

- 14 -
2.4 Topic Maps

2.4.1 Introduction

2.4.1.1 Topic Maps

Topic Maps are defined as an ISO standard, ISO/IEC 13250 [19], which conforms to a specific syntax defined in SGML and HyTime [16, 44, 48]. The standard includes, as an annex, XTM 1.0, a syntax defined in XML in order to make Topic Maps relevant for the Web by defining a more specific Topic Map model [25].

Topic Maps fulfil the same role as a back-of-book index – managing vast quantities of data for navigation and retrieval by associating reified topics within an information layer with one another as well as with real world subjects [6, 16, 44, 52, 55]. This is accomplished by creating an abstracted information layer without altering the original data [48, 54]. Embedded within the syntax are complex mechanism providing for the merging of indexes that aid in combining like subjects in an overall information model [48].

The syntax is further extended by a Topic Map Query Language (TMQL) and a Topic Map Constraint Language (TMCL) [48].

2.4.1.2 Application of Topic Maps

In the age of infoglut, it has become a major challenge to find desired information in a timely fashion [6]. The Topic Map description of information solves this problem by allowing for the simple navigation of an information layer that is abstracted from the underlying data in a concise and intuitive manner [6, 44, 48]. Navigation is accomplished by following associated links between reified topics relating to real world subjects of interest [52, 55]. This allows for users to get a quick overview of a desired subject. If specific questions are known, a query may be specified in order to extract the relevant information [48].

Navigating, querying or generating Topic Maps have no impact on the underlying data and therefore allow for a safe channel of interaction with real world subjects without the risk of loosing or altering any information [44, 55].

2.4.2 Background

2.4.2.1 History
In 1991, the founding of the Davenport group under the leadership of Steven R. Newcomb resulted in the beginning of work on Topic Maps [38, 39]. This initiative, known as SOFABED (Standard Open Formal Architecture for Browsable Electronic Documents) studied how the HyTime standard could be used for the representation of interoperable indexes [7].

Steve Newcomb later admitted in a private communication to Steve Pepper that his own views were regarded as futuristic and that in 1993 the group split in two [40]. One group continued to create DocBook while Newcomb remained as the convenor for the second – Conventions for the Application of HyTime (CApH) operating in the context of GCA Research (now known as IDEAlliance) [7, 38].

Michel Biezunski joined Steve Newcomb at CAPH and in 1995 they co-edited the basic model of Topic Maps. At that stage the model was mature enough for the ISO/JTC1/SC18/WG8 workgroup to accept it as a new work item – a proposed standard [7].

The Topic Map standard was ultimately published in 2000 as ISO/IEC 13250:2000. As soon as the ISO/IEC 13250:2000 standard was published work began on the XTM initiative in order to make Topic Maps relevant for the web [38]. This initiative, an independent organisation known as TopicMaps.org, was chaired by Michel Biezunski and Steven Newcomb [7, 38].

XTM 1.0 was published in 2000. In 2001 it was slightly amended and by 2002 had been added as an annex to ISO/IEC 13250:2000 [38]. Steve Pepper and Graham Moore were appointed as editors of the new standard and Eric Freese was appointed as chair of TopicMaps.org [7].

2.4.3 Description

2.4.3.1 ISO Topic Maps

ISO Topic Maps were first published in 2000 as ISO/IEC 13250:2000 [16, 25]. The specification makes use of syntax in SGML and HyTime [44]. After publication, three problems were identified with ISO Topic Maps [34]:

- The syntax was defined in SGML with no XML
- HyTM was used for linking with no URI’s
- The syntax used architectural forms, so there was no fixed syntax

Standard Generalised Markup Language
The Standard Generalised Markup Language (SGML) was published 1986 as ISO/IEC 8879:1986. SGML came about due to a need for describing documents in terms of their logical structure [34].

SGML contains two parts [34]:
- A metasync for expressing individual document types known as a DTD (Document Type Definition)
- A metasync for describing the markup in the documents themselves

Documents described using SGML may be validated by a single validating parser which is tasked to decide if a specific document conforms to a DTD in question [34].

**HyTime**

The Hypermedia/Time-based Document Structuring Language is an extension of SGML specified by ISO/IEC 10744 [9, 10, 34]. HyTime allows all documents to package their content using standard markup[34]. This markup allows for the interpretation of notations regarding the content of the document by any application [34].

HyTime comprises of a DTD (Document Type Definition) of SGML elements each of which has a specific meaning [9]. HyTime is further comprised of several modules to allow for the scheduling and location of specific documents [34].

**2.4.3.2 XTM 1.0**

The problems prevalent with ISO Topic Maps were identified soon after publication and an independent consortium, TopicMaps.org, was formed to define a web-enabled syntax [25]. This syntax was included as an annex to the ISO/IEC 13250 specification as XTM 1.0 and included a syntax defined in XML as well as using URI’s for linking [42].

**XML**

XML consists of a way to define data formats, a way to communicate details about the data formats and a style for formatting data. XML uses embedded tags to separate content from format in order to describe the structure of a document [53].

XML was inspired by SGML and therefore contains a DTD SGML subset for pre-defining document structures [53].

XML documents may follow a hierarchical structure with elements containing child attributes. Such attributes may be user-defined and specified in a DTD intended for that specific document class [53].
2.4.4 Topic Maps Syntax

The purpose of the Topic Map syntax is to create a superimposed layer of information metadata identifying a collection of related subjects of interest [6, 48]. This layer fundamentally consists of Topics, Occurrences and Associations all linked together in order to form an intuitive and navigable representation of underlying data [6, 25]. Such data may be stored virtually anywhere, such as a spreadsheet, databases, webpages or the Topic Map itself and is pointed to by URI’s within the Topic Map.

![Figure 5: A Simple Topic Map](image)

This allows for the complete integration of data from any source by creating an abstracted layer of information regarding that data and its exact location [42]. This layer exists outside of the data in question and may be altered or queried without affecting the underlying data or data dependent on it [55].
Figure 5 above presents a topic map representation of the UML diagram shown in section 2.2. This topic map contains exactly the same information, however, is a better visual representation due to the ability to name the relationships between topics.

In addition, information may be stored in locations other than the topic map itself. This functionality allows for a powerful layer of metadata which may reference information stored in a number of non-central locations.

### 2.4.4.1 Topics

At the heart of all Topic Maps is the concept of a topic [16, 33]. A topic is the reification of a subject in the real world and can be used to represent absolutely anything, such as an object, a person or an idea [29, 42, 55].

Topics are linked to each other via associations in order encode additional knowledge about a topic in a specific topic space. This form of linking allows for the intuitive navigation of topic maps [42].

Each topic may have multiple occurrences and names [16, 42]. A topic may be typed, allowing topics to be categorised according to their kind. Typing a topic consists of inheriting from a parent topic using an <instanceOf> relationship [6, 48].

### 2.4.4.2 Occurrences

As previously mentioned, each topic may have several occurrences. Occurrences allow for data to be added to the topics that they identify [42].

Each occurrence is typed using an <instanceOf> relationship. Such an occurrence type may be attributes such as date-of-birth or information such as topic descriptions [6, 42].

Each occurrence contains a <topicRef> value linking to the desired data in question. This data may be stored within the Topic Map itself or point to data stored elsewhere with the use of URI’s [29].

Since occurrences are linked to the topic itself, any person with access to that topic would automatically have access to the related information or data provided by its occurrences [42].

### 2.4.4.3 Associations

Associations allow for topics to be linked together within a certain context. Associations describe a multi-directional relationship between related topics defining each topic as a member that plays a specific role in that association [5, 29, 48].
Associations are also bi-directional [52]. Thus if an association exists indicating that Shakespeare wrote Hamlet, it automatically follows that Hamlet was written by Shakespeare. This may seem intuitive, however, the ability to extrapolate relationships from Topic Map definitions allow for powerful applications [52, 54, 55].

An association may define a relationship between two or more topics by adding each topic as a member <member> in the association and giving it a role <roleSpec>. Such association roles are modelled by using topics to define the role in question [6, 55].

Associations too may be typed by defining an <instanceOf> relationship in order to inherit from its parent. This association typing allows for associations to be categorised together and is of use when extracting information from topic maps [6, 48].

### 2.4.4.4 Merging

A powerful feature of Topic Maps is the ability to merge maps together. Topic Maps may be merged if both have topics that refer to the same subject [42].

When Topic Maps are merged a union of the similar topics is created consisting of all names, occurrences and associations that were present in the original topics. Such merging works best when topics refer to identical subjects producing a meaningful set of new information [42]. In this way multiple views of the same information may be merged together to create a complete overview of the underlying data.

With Topic Map merging, eventually, thousands of Topic Maps may be merged together in order to create a vast index of data. It is this functionality that lead Charles Goldfarb to describe Topic Maps as “the GPS of the information universe” and seems to fulfil Tim Berners-Lee’s vision of the semantic web – a web were all data is linked together allowing understandability by humans and machines alike [30, 52].

### 2.4.4.5 Topic Map Constraint Language

The Topic Map Constraint Language (TMCL) was designed to constrain any aspect of a Topic Map. TMCL is currently an ISO/JTC1/SC18/WG8 work item and is planned to be included to the Topic Map family as ISO/IEC 19756 [21, 29].

TMCL makes use of the Topic Map Query language (TMQL) to specify both the aspects of the Topic Map to be constrained as well as the Topic Map structures that must be in place for all constraints to be met [21].

TMCL consists of three distinct parts [21]:

- Predicates
The predicates define which TMDM (Topic Map Data Model) construct is to be constrained. The constructs used to describe these predicates are comprised of several core TMQL constructs that can be combined in various ways to form complex expressions [21].

The constraint constructs use the above mentioned predicates to complete the application of constraints on the TMDM [21].

Finally, the TMCL-Schema describes a language that can be used to constrain topics and associations [21].

The implementation of the TMCL as described above allows Topic Map authors not only to define information layers of real world data, but to define a way in which these items may relate to each other [21].

A powerful use of this constraint language concerns the application of topic maps to a specific ontology within an organisations framework. Defining a constraint language to allow for the implementation of certain business rules allows for automatic compliance of the associated rules as well as reusability should individuals responsible for constraints no longer be available [48].

### 2.4.4.6 Topic Map Query Language

The purpose of TMQL is to create a standard query language for Topic Maps. TMQL is currently an ISO/JTC1/SC18/WG8 work item and is planned to be included to the Topic Map family as ISO/IEC 18048 [20, 29].

To date a requirements document as well as a use case document for TMQL has been published with the first working draft published in February 2005 [20].

Since TMQL is not yet a published standard, it is still in the process of refinement and subject to changes, however, at present consists of three sub-languages [20]:

- Path Expressions
- SELECT Expressions
- FLWR Expressions

Path expressions are very much like XPath expressions, starting with a value and generating subsequent values in steps under a specific set of conditions [20].

Select expressions or predicates operate very much like tolog, a prolog like querying language following a SQL like syntax [20, 48].
FLWR expressions are inspired by XQuery and the way it uses XPath and add even more power to the querying language [20].

2.4.5 Implementations

Despite the relatively new status of the Topic Map specification, a wide variety of commercial and open-source solutions are already being offered that make use of the power of Topic Maps.

All of the implementations have certain features in common such as an API to retrieve and manipulate data.

2.4.5.1 Software Engines and Visualisation

TM4J

TM4J is an open source project consisting of an API for developing processing and exporting topic maps [48, 1].

TM4J supports different storage providers such as in-memory or relational databases. Further, TM4J includes an implementation of the tolog query language for Topic Maps [48, 1].

TM4J allows for the run-time merging of Topic Maps as well as functionality to harvest topics from certain formats.

A subproject of TM4J, know as TMNav is currently in development. TMNav, another open source tools makes use of the TM4J libraries to allow a user navigable interface for Topic Map visualisation.

Ontopia Omnigator

The Ontopia Omnigator is a web-based Topic Map visualisation tool built on the Ontopia Navigator Framework allowing the navigation, visualisation and querying of XML Topic Maps [37].

The Omnigator supports a large variety of syntaxes, allows for the merging and exporting of Topic Maps and includes semantic validation [37].

The GooseWorks Topic Map Toolkit

The GooseWorks Topic Map processing engine provides an API written in C with a wrapper API in Python [1].
The Toolkit provides persistent storage in relational databases as well as a command line tool for file operations such as exporting and merging [1].

**SemanText**

SemanText is an open-source semantic network application developed in Python. The application uses Topic Maps to store and manage knowledge [1].

SemanText has the functionality to harvest topics from XML and RDF files [1].

2.4.5.2 **Known Topic Map Implementations**

**Publishing Solutions**

**The IRS Tax Publications**

The United States Inland Revenue Service (IRS) have developed a Topic Map indexing system to solve their problems with unifying and merging large volumes of tax publications [1].

**Web application Development**

**Patrimoine Financial Documentation**

Patrimoine, a provider of professional and legal documentation currently make use of the Intelligent Topic Manager (ITM), provided by Mondeca, to keep track of all topics and indexes on those topics [1].

The ITM makes use of a topic map graph to represent the consolidated information [1].

**Application Development**

**Bravo – Knowledge Management with Topic Maps**

GlobalWisdom Inc. has developed Bravo, a knowledge management solution that uses Topic Maps to create scalable information architecture [1].

Bravo uses the K-43 Topic Map Engine from Empolis [1].
Chapter 2: Background

Application Integration

Starbase

Starbase Corporation is making use of Topic Map technology from Ontopia in a new product architecture that focuses on extracting value from information [1]. This architecture will allow end users to manage, navigate and query underlying data in order to extract information of value from large repositories of information [1].

2.5 Background Summary

A critical aspect of software development is that of requirements management. Without successful management of requirements by using concepts such as requirements traceability the likelihood of project failure increases greatly.

This chapter introduced the goals of requirements management and requirements traceability with an aim of showing the importance of managing requirements during system design.

The Unified Modelling Language (UML) is briefly discussed. UML provides a visual modelling language and can be used to model a large variety of objects. UML may provide solutions to the requirements traceability problem by allowing for visual modelling of requirements to allow for intuitive change impact analysis and monitoring – a problem that is thoroughly discussed in section 2.3 above.

A possible improvement on the visual modelling of UML is presented in the form of XML Topic Maps (XTM). Topic maps provide a syntax for modelling objects and the relationships between them – similar to that of UML. An advantage of XTM, however, is the ability to logically model, visualise and navigate a data aware structure. The ability to merge two topic maps, as described in section 2.4 gives a possible solution to the problem of merging requirements traceability as discussed in section 2.3.

A further benefit is the ability to filter and query XTM allowing for possible techniques to automate the impact of changes in requirements traceability – a problem introduced and described in section 2.3.
Chapter 3: System Design

This chapter aims to give the reader an overview of the design process followed in creating a system for supporting the central concepts of this thesis. In this thesis, the Rational Unified Process and its application herein is introduced. The following design is thoroughly discussed based on requirement elicitation, use-case modelling and package design. Finally, a comprehensive discussion of certain design decisions is presented.

3.1 Rational Unified Process

3.1.1 Overview

The Rational Unified Process (RUP) is a software engineering process that manages the development of a software project by assigning relevant tasks and processes within a development organisation [45].

Figure 6 above shows the four phases of which RUP comprises. The Horizontal access indicates project time, while the vertical access shows the disciplines within RUP. These disciplines logically group the nature of activities within a software development organisation together [45].

Phases

RUP consists of four distinct phases. Each of these phases can be iterative in nature and occur as time progresses:
Inception
The goal of the inception phase is to complete a conceptual vision of the planned system by eliciting requirements from all stakeholders involved. By the end of this stage, there should be a complete conceptual plan along with assessed feasibility in order to proceed. In smaller sized projects, there is usually only one of these inception phases.

Elaboration
The primary goal of the elaboration phase is to build a solid framework for the upcoming construction phase. It is in this phase where a proven architecture is developed based on the most significant system requirements.

Construction
The construction phase concentrates on completing system development based on the architecture framework created during the elaboration phases. The goals of the construction phase are to complete stakeholder requirement elicitation and complete the development of the system.

Transition
The transition phase focuses on releasing a working system, testing and refining the system based on user needs. There can be as many transition phases as needed during the RUP lifecycle.

Disciplines
The Rational Unified Process defines various disciplines play roles in all aspects of software development organisations. These disciplines all run concurrently during any one of the mentioned phases. Of these disciplines, only the following are of interest for the scope of this thesis:

Requirements
The requirements discipline focuses on the elicitation of stakeholder requirements. This discipline constitutes workflows for analyses of the system, requirements elicitation, system definition and management and requirements change management.

Analysis and Design
The analyses and design discipline focuses on defining the system architecture, developing components and architecture refinement based on the stakeholder requirements elicited.

Implementation
The implementation discipline focuses on the implementation and construction of the components designed. Implementation also focuses on the integration of the developed components.
• **Testing**  
The testing discipline, as the name implies, concentrates on aspects of testing the system throughout development as well as refining the test plan after each iteration.

• **Project Management**  
Project management is the discipline that oversees the development process. Project management achieves this by firstly being responsible for project conception and then managing the project throughout development. Project management is responsible for planning iterations and evaluating project scope and risk.

### 3.1.2 Application

In this project, the Rational Unified Process was used as a process to oversee the system development process of the conceived project. RUP was tailored in order to suite the size of the project, the size of the team as well as time constraints in place.

Below is a brief description of the tailored process used for system development.

**Description**

Prior to tailoring RUP, a preconception phase was completed. This preconception phase, as detailed in section 3.2, allowed for concept verification initially and during the RUP conception workflows.

The first phase of project development consisted of tailoring the Rational Unified Process to suiteing the specific needs and goals of the project given the known constraints.

<table>
<thead>
<tr>
<th>Table 1: RUP Disciplines and Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discipline</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
The disciplines in table 1 were selected as being central to system development and meeting the specific requirements given the limitations of system size, project team size and time constraints. Along with each discipline are the most critical discipline specific workflows that were followed throughout system design. The workflows were further refined in order to provide an agile development process. Each workflow was selected based on how it would contribute to the goals of the development process.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>40 days</td>
</tr>
<tr>
<td>Inception</td>
<td>7 days</td>
</tr>
<tr>
<td>Elaboration</td>
<td>14 days</td>
</tr>
<tr>
<td>Construction</td>
<td>15 days</td>
</tr>
<tr>
<td>Iteration 1</td>
<td>7 days</td>
</tr>
<tr>
<td>Iteration 2</td>
<td>7 days</td>
</tr>
<tr>
<td>Iteration 3</td>
<td>5 days</td>
</tr>
<tr>
<td>Transition</td>
<td>4 days</td>
</tr>
<tr>
<td>Iteration 1</td>
<td>4 days</td>
</tr>
</tbody>
</table>

**Figure 7: Iterative Development Timeline**

All four of the phases were followed during the development process. Figure 7 shows the timeline that was followed during development. As can be seen, the development process consisted of:
- Inception Phase
  Only one inception phase was followed. This phase allowed for the elicitation of the stakeholders requirements as well as the development of the conceptual framework for the system.

- Elaboration Phase
  Two elaboration phases were needed as requirements changed and risks were mitigated. The two elaboration phases allowed for the complete development of a system design allowing for construction.

- Construction Phase
  Three construction phases were needed due to iterative testing, risk mitigation and requirement elicitation. The construction phases allowed for the development of a final system.

- Transition Phase
  A single transition phase allowed the further refinement of the system based on user feedback.

The following sections in this chapter will provide an overview of the final system design based on the RUP iterative development process. Appendix A may be consulted for a detailed account of all iterations.

3.2 Preconception

The preconception phase centred on the concept of requirements traceability modelled using UML. Requirements and their trace dependencies where modelled in UML and the requirement traceability was demonstrated by means of a traceability matrix.

3.2.1 UML Modelling of Requirements

Requirements were modelled based on Justin Kelleher’s PhD related research [22] to date. Requirements were modelled as UML class diagrams with each requirement given specific attributes such as risk, cost and priority. Stereotypes were assigned to each requirement class in order to define its type – functional or non-functional.

Traceability links were modelled using UML relationships (see: Section 2.2) between requirement objects.

Figure 8 shows the results of modelling requirements traceability using this approach. In this figure, there are three requirements – ReqA, ReqB, ReqC. Each requirement has an associated cost, risk and priority modelled as attributes.
A traceability link exists between ReqA and ReqB and is defined by a UML aggregation relationship indicating that ReqB uses ReqA. Thus, due to the traceability between ReqA and ReqB, any change in ReqA would impact on ReqB.

### 3.2.2 Requirements Traceability Matrix

Another approach to displaying requirements traceability is the use of a traceability matrix. A traceability matrix shows a two dimensional view of requirements plotted against one another. Traceability links may be defined by specifying a trace-to or trace-from relationship between selected requirements in a specific cell.

Figure 9 shows the results of placing ReqA, ReqB and ReqC in a traceability matrix. In this figure, there are once again three requirements each displaying the particular traceability link as seen in the UML class diagram example.

Unfortunately, due to simplification of the model, information has been lost regarding the type of traceability link.
3.2.3 Exporting UML Models

In order to allow for visual representation of UML diagrams as XML Topic Maps, an automated way needed to be found for exporting and importing of models. This project’s scope covers the exporting of models to a suitable format to be used for conversion to XML Topic Maps.

The eventual export format selected was the XML Metadata Interchange syntax (XMI). XMI allows for a portable easy to navigate data container.

3.3 System Requirements

The process of requirements gathering was conducted primarily during the inception phase of the project. Since there was only one inception phase, the requirements specification was refined during the two elaboration phases due to the iterative nature of RUP.

This section highlights the final requirements as well as the evolution of additional requirements during the various iterations of RUP.

3.3.1 Requirements Elicitation

Functional and non-functional system requirements were gathered throughout all phases and iterations of system development using RUP. These requirements were analysed and verified in order to ensure the system’s conformance to the stakeholders’ goals. Considerable emphasis was placed on research in section 2 in eliciting requirements.

3.3.1.1 Functional Requirement Elicitation

Functional requirements were elicited using frequent requirement workshops. These workshops allowed for an environment of brainstorming with the stakeholder in order to elicit optimal system requirements.

The following high-level functional system requirements of the system appeared during development and were the final functional requirements behind driving development and the final system:

- The user must be able to create a new project
- The user must be able to open an existing project
- The user must be able to manage requirements
- The user must be able to manage traceability links
3.3.1.2 Non-functional Requirement Elicitation

Although the task of assessing non-functional requirements was greater, frequent meetings and design sessions lead to the following non-functional requirements:

- The user interface must be intuitive
- The user interface must appear professional
- Requirements and attributes must be added seamlessly
- Traceability links must be added seamlessly

Due to the scope of the project, not much emphasis was place on the non-functional requirements opting for a solution to support the projects aims rather than system usability.

3.3.1.3 Requirements Traceability

Figure 10 indicates an overview of the traceability existing between high-level user requirements for the system to be designed. Creating a project traces to the opening of a project, since no project can be opened if it has not yet been created. Both of these requirements trace to the adding and editing of requirements. Finally, Adding and editing of requirements traces to the adding and editing of traceability since traceability cannot be managed without existing requirements.

![Figure 10: Traceability of Requirements](image)

3.3.2 Use-case Modelling

Functional requirements were modelled as use-case diagrams in order to aid in the design process. Two high-level requirements were identified from the given set of functional requirements:

- The user must be able to manage projects
- The user must be able to manage requirements traceability

The high-level requirements can be understood through the use-case diagram shown in figure 11 below.
The use-case shows that users must be able to manage projects, such as open and close files, as well as manage requirements traceability, such as managing requirements and their attributes as well as traceability links.

### 3.3.2.1 Project Management Requirements

One of the primary requirements for this project is that users must be able to manage projects that they are working with. This requirement can be separated into two of the four functional system requirements, create project and open project. Based on the research in Chapter 2 further low-level requirements based on modelling system functionality became evident. These are defined beneath their requirements below:

- **Users must be able to create projects**
  - A project name must be selected
  - User access must be definable
  - Functional and non-functional attribute types must be definable
  - Relationship types for traceability links must be definability

- **Users must be able to open projects**
  - The user must be able to select a project to open
  - The user must enter a user name to gain access

Figure 12 shows a use-case diagram of the project management requirements.
Requirements for project creation and project opening are discussed below.

**Project Creation**

In order to create a project, the user must be able to define a project name, enter in her name as project creator, define an access list, define functional and non-functional requirement attribute types and define a set of associations or relationship types.

Figure 13 shows a use-case diagram of the project creation requirements.

In the context of project creation, the user plays the role of the Project Creator. The project creator can create a new project by specifying a project name and a project creator.
name. The project creator can also specify an access list, functional and non-functional requirement attribute types and associations. Each activity is dependent on the ability to enter text. Creating a project also saves a file to the project directory.

Thus, the requirements extracted for the project creation requirement are identical to the stakeholders’ requirements, with one addition:

- User must be able to enter a project name
- User must be able to enter a project creator name
- User must be able to specify an access list
- User must be able to enter functional and non-function requirement attribute types
- User must be able to specify association types
- A new project file and topic map file must be created upon project creation.

**Opening a Project**

In order to open a project the user must be able to specify a project name and a user name.

Figure 14 shows a use-case diagram of the project opening requirements.

![Figure 14: Open Project Use-case](image)

In the context of opening a project there are two actors – the project creator and a project user. Both actors can specify a project to access along with a user name with which to gain access. Opening a project is dependent on loading a project file, containing setup information, and a topic map file, to display, from the project directory.

Thus, the requirements extracted from the project opening requirement are identical to the stakeholders’ requirements with one addition:
- User must be able to specify a project name
- User must be able to specify an access name
- A project file and a topic map file must be loaded from the selected project directory.

### 3.3.2.2 Requirements Traceability Requirements

The second primary requirement of the system is that users must be able to manage requirements traceability. This requirement can be divided into the remaining two original high-level stakeholder requirements. Based on research from Chapter 2, additional low-level stakeholder requirements have been added. These are displayed below each of the high-level requirements.

- **Users must be able to manage requirements**
  - Users must be able to add and edit requirements
  - Users must be able to add and edit attributes
- **Users must be able to manage traceability**
  - Users must be able to edit traceability links

Figure 15 shows a use-case diagram of the requirements traceability requirements.

![Figure 15: Requirements Traceability Use-case](image)

Requirements for managing requirements and managing traceability are discussed below.

### Managing Requirements

In order to manage requirements a user must be able to add and edit requirements and attributes associated with those requirements by defining properties for each of the edited entities.

Figure 16 shows a use-case diagram of the requirements management requirement.
In the context of managing requirements there is only one actor – a general user. This general user can be either a project creator or a project user. The general user can add and edit requirements by defining new values for them as well as add and edit attributes. The user can also select a topic map file to edit and save the file to disk. Entering in details requires the ability to enter text.

Thus, the requirements extracted from the requirements management requirement are identical to the stakeholders’ requirements with several additions:

- Users must be able to add and edit requirements
- Users must be able to add and edit attributes
- Users must be able to select a topic map to edit
- Users must be able to save or cancel changes

**Managing Traceability**

In order to manage traceability a user must be able to edit traceability links between requirements.

Figure 17 shows a use-case diagram of the traceability management requirement.
In the context of managing traceability there is only one actor— a general user. This general user can be either a project creator or a project user. The general user can edit traceability links by specifying a new value. Entering values requires the ability to add text.

Thus, the requirements extracted from the traceability management requirement are identical to the stakeholders’ requirements:

- Users must be able to edit traceability links

### 3.3.2.3 Traceability of Use-cases

Figure 18 indicates an overview of the traceability between high-level requirements and use-cases described above. Each high-level requirement traces to one use-case.
3.4 System Overview

The initial system concepts were defined and envisioned during the single inception phase, however, a predominant amount of analysis and design effort was spent during the two iterative analysis and design phases of RUP. This iterative process allowed for the simultaneous refinement of requirements and system design artifacts.

Presented over the rest of this chapter are the final designs that were yielded after the development of the system.

3.4.1 High Level Overview

The high level overview of the system is a conceptual model of critical components and data flows throughout the system. The system essentially contains 8 core components and 5 data flows.

Figure 19 illustrates this system overview.

![Figure 19: High-level System Overview](image)

The core component in the systems architecture is that of the TMNav visualisation tool. This tool is used to display requirements traceability using XML Topic Maps.

- 39 -
Components dependent on TMNav are the Requirements Traceability Management component and the File Management component.

Information is imported into Topic Map Format (XTM) from XMI. This is loaded into TMNav using the File Management Component. The Requirements Traceability component is used to edit the XTM file which can then be saved.

### 3.4.2 Low Level Overview

The low level overview is that of a UML package diagram illustrating the core packages or meta-packages involved with the systems functionality.

Figure 20 illustrates the low-level system overview.

![Figure 20: Low-level System Overview (Packages)](image)

The core of the system is TMNav which allows the display and navigation of XML Topics Maps. TMNav uses libraries from TM4J, a topic map engine, as well as Panckoucke, a package for loading topic maps into a displayable data structure.

The user interface is built into TMNav and allows for integration. The file manager, requirements manager and traceability manager are both integrated into this interface.
The traceability manager is launched from the requirements manger and any changes made to traceability links are saved via the requirements manager.

Both the file manager and requirements manager are dependent on the parser for loading and saving topic maps. The parser in turn requires JDOM, the Java Document Object Model for accessing XML files to traverse the topic map.

Each interfacing system component is explained in detail in the following sections.

### 3.5 File Manager

The file manager is used to manage the creating and loading of project files with associated topic maps for the system.

Figure 21 below shows the class diagram and high-level dependencies of this package.

![File Manager Package](image)

**Figure 21: File Manager Package**

Operations and attributes have been omitted from this model for simplicity. The file manager contains two bases classes.

- The projectCreator (Creator) class
- The projectOpener (Opener) class
3.5.1 Project Creator

The Creator is responsible for allowing a user to create a new project. This is accomplished by allowing the user to define a new project name, a creator name, an access list, functional and non-functional requirement attribute types and association types as specified by section 3.3.

Additionally, the Creator creates a new topic map file as well as a new project file containing all the specified information as specified by additional requirements in section 3.3.

3.5.2 Project Opener

The Opener is responsible for allowing a user to open an existing project. This is accomplished by allowing the user to select an existing project and an access name as specified by section 3.3.

Additionally, the Opener loads the project information as well as opens the correct topic map for viewing as specified by additional requirements in section 3.3.

3.6 Parser

The parser is responsible for loading a selected topic map file into a traversable data structure by walking through a DOM tree of the topic map and finding all requirements, attributes and associations. Further to this, the parser saves edited requirements, attributes and traceability data structures back to the topic map.

Figure 22 shows the class diagram and high-level dependencies of this package.

![Figure 22: The Parser Package](image-url)
The parser package consists of the core parser class, a requirement class for storing requirements, an attribute class for storing attributes, an association class for storing associations and a member class for storing association roles.

### 3.7 Requirements Manager

The requirements manager is responsible for allowing the adding and editing of requirements as well as the adding and editing of attributes.

Figure 23 shows the class diagram and high-level dependencies of this package.

![Figure 23: The Requirements Manager Package](image)

The requirements manager consists of a requirements management user interface which allows for the adding and editing of requirements and attributes as specified by section 3.3. Adding and editing is accomplished by allowing the user to enter values for each requirement or attribute.

Additionally, the requirements manager allows for the selecting of the topic map to be edited as well as the saving of the topic map once edited as specified by the additional requirements in section 3.3.

The traceability manager, which will be discussed under the next section, is also launched from the requirements manager.
3.8 Traceability Manager

The traceability manager is responsible for allowing the viewing of traceability links as well as the editing or adding of traceability links.

Figure 24 shows the class diagram and high-level dependencies of this package.

![Figure 24: The Traceability Manager Package](image)

The traceability manager is launched from the requirements manager and uses the imageTable class to show a traceability matrix as well as allow for the editing of traceability links as specified by section 3.3.
Chapter 4: Implementation

After initial system design, the construction of the system began. Since the RUP framework was followed, the process was strictly iterative progressing through three construction phases involving implementation, testing and redesign based on changing user requirements and risk mitigation.

This section presents the final system implementation as described by the system designs created during the elaboration phases. The aim of this chapter is to show the results of the final system implementation and its conformance to the original design documentation.

4.1 File Manager

The final implementation of the file manager very closely follows the envisaged designs laid out in Chapter 3. The project creator as well as the project opener where both implemented as detailed and the implementation of both is described below.

4.1.1 Project Creator

The project creator allows the user to create a new project as outlined in Chapter 3. Text boxes where used for inputs of all values such as selecting the project name, selecting the project creator, entering access right, entering attribute types and entering association types.

Figure 25 shows the final implementation of the project creator GUI.

![Figure 25: Final Project Creator Interface](image-url)
The project creator GUI was designed using Java Swing components. Further, the project creator fulfils additional requirements highlighted in Chapter 3 such as creating a new project file containing project specific information as well as creating a new topic map file.

4.1.2 Project Opener

The final implementation of the project opener allows the user to open an existing file as outlined in Chapter 3. Comboboxes where used to select existing projects in the projects directory. Upon selection, the access list is updated with user names that can be selected using a combobox.

Figure 26 shows the final implementation of the project opener GUI.

![Figure 26: Final Project Opener Interface](image)

The project opener GUI was implemented using Java Swing components. Further, the project opener fulfils the additional requirements as laid out by Chapter 3 such as loading the selected project file as well as the associated topic map file from the projects directory.

4.2 Parser

The parser was implemented according to the class diagram depicted in Chapter 3.

Two separate vector data structures were created to hold requirements with their attributes and traceability links.
The requirement vector holds requirement classes for storing requirement information. Each requirement contains a vector of attribute classes that store attributes and their details.

The traceability vector holds trace classes for storing traceability relationship information. Each trace class contains a member vector that stores relationship members and their roles. A member vector was used in this instance as it was foreseeable that the system may need to be adapted for the definition of one to many relationships in the future.

The requirement vector and the traceability vector are linked together by means of the traceability vector members’ relative position in the requirement vector. This was done in order to keep the updating of traceability details as simple as possible.

### 4.3 Requirements Manager

The requirements manager was implemented as described in Chapter 3. The requirements manager allows for an intuitive interface to add and edit both requirements and their attributes.

Figure 27 shows the final implementation of the requirements manager GUI.
The requirements manager allows for the addition or editing of requirements. Editing the requirements values is accomplished by means of textual fields for names and values as well as comboboxes for selecting requirement or attributes types from the predefined attribute type lists specified by the user at project creation.

4.4 Traceability Manager

The traceability manager was implemented as described in Chapter 3. A traceability matrix was created to display traceability links between requirements and allow for the editing of traceability links between requirements.

Figure 28 shows the final implementation of the traceability manager GUI.
Figure 28: Final Traceability Manager Interface

As can be seen from Figure 28, the traceability matrix shows a visual representation of traceability information. The matrix contains links from parent requirements shown vertically, and their children, shown horizontally.

The editing of traceability links is accomplished by a combobox that drops down when the user clicks a specific cell. The combobox contains a list of association types as specified by the project creator at the time of project creation.

4.5 Integration and Final System

4.5.1 Integration

All components were integrated into TMNav and its user interface at the end of every iteration in which development of components occurred. This was accomplished as a result of the iterative nature of the Rational Unified Process.

During implementation, the development of all components was kept very modular as defined by RUP’s component-based construction framework. This modularity allowed convenient integration with the various packages forming TMNav.

Certain problems were faced during system integration. These problems are highlighted below:

- Integration with TMNav’s core file handling functions were problematic due to a lack of design documentation and code comments.
- TMNav’s heavy integration with the Panckoucke library made for a highly complicated layer of code, resulting in the need for careful integration with affected modules.
4.5.2 Final System

The final systems allows for the user to achieve the following tasks as specified by Chapter 3:

- Create a new project
- Open an existing project
- Manage requirements
- Manage traceability

Figure 29 shows the final implementation of the system.

![Figure 29: Final System Interface](image)

The system contains a menu structure at the top to allow for navigation between essential functions. Further, the system contains a panel that displays the opened topic map file associated with the current project. The indexes view on the left indicates all topics in the topic map and can be refreshed to show information such as the associations and roles.

The main panel is split into two parts. The first is a tree view displaying an overview of the currently selected topic. The second, on the right, is the main topic map panel, responsible for displaying the topic map and allowing for navigation.
Users can navigate the requirements traceability topic map by clicking on the requirements that they wish to view. Users can also retrace their steps by click the forward and back browser buttons at the top of the main panel.

The topic map itself displays the requirements, attributes and associations. Requirements are defined by a green ‘T’ icon, attributes are defined by a black ‘O’ icon and associations are defined by a textual string marking the link between requirements. Each traceability link is also contains a textual string indicating the role that the requirement plays in the relationship, such as parent or child.

The existing functionality of the TMNav development base adds further functionality to the system:

- Viewing requirements traceability using topic maps
- Navigation of requirements traceability contained within topic maps
- The merging of topic maps at run time to allow for the combination of requirements specifications for a single project.
- The ability to visualise requirements traceability using various render views such as:
  - The Touch Graph Render View
  - The Hyper Graph Render View
  - A Tree View
  - A Table View
- The ability to modify the various displays of the render views using abstractors such as:
  - Compact abstractor for a compact view
  - A role based abstractor for placing the focus on associations
  - A section abstractor for view a logical representation of the topic map
  - An extended abstractor for viewing complete topic map information

### 4.6 Traceability of Implementation

Figure 30 indicates an overview of high-level requirements traced to use-cases which then trace to their specific implementations as described above. Each use-cases trace to a specific implementation.
Figure 30: Traceability of Implementation
Chapter 5: Testing and Validation

This chapter aims to inform the user of testing techniques and their implementation throughout the thesis. Techniques are introduced and explained thoroughly. The resulting insights gained from implementations of these techniques are thoroughly discussed throughout subsequent chapters.

5.1 Testing Techniques and Types

5.1.1 Testing Methodology

Figure 31 above indicates a hierarchical structure of the three levels of testing conducted.

At the lowest level is testing regarding functionality traced to use-cases from Chapter 3. This functional testing of use-cases and therefore stakeholder requirements of the developed system provides feedback regarding implementation success.

At the middle level is testing regarding the validation of design decisions and other output data. This form of testing will allow the validation of the various design decisions made as well as the validation of the XML Topic Map files generated.

At the highest level of testing there is proof of concept testing. This is the testing of the concept implemented within this thesis and regards its success and validity. This form of
testing will provide feedback regarding the concept discussed and implemented in this thesis with regards to validity and success.

The following testing techniques were taken from the RUP test plan created by the project members:

5.1.2 Functional Testing

<table>
<thead>
<tr>
<th>Table 2: Functional Testing Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique Objective:</td>
</tr>
</tbody>
</table>
| Techniques: | ▪ Xtreme Programming Acceptance tests  
              ▪ User Tests with metric |
| Oracles: | ▪ Programmers evaluation of Acceptance tests  
            ▪ The metric for the user test has a goal that minimum percentage of users can perform the function. |
| Required Tools: | |
| Success Criteria: | Evaluated functions pass requirements for both the acceptance tests and the metric. |
| Special Considerations: | |

5.1.3 XTM Validation Testing

<table>
<thead>
<tr>
<th>Table 3: XTM Validation Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique Objective:</td>
</tr>
</tbody>
</table>
| Technique: | Generated Topic Maps are tested on the XML validation forms provided by:  
              ▪ the Scholarly Technology Group at Brown University:  
              http://www.stg.brown.edu/service/xmlvalid/  
              described at:  
              http://www.stg.brown.edu/service/xmlvalid/Xml.tr98.2.shtml  
              referenced by:  
              http://www.xml.com/pub/r/510  
              ▪ Language Technology Group at the University of Edinburgh.  
              described at:  
              http://www.hcrc.ed.ac.uk/~richard/xml-check.html  
              referenced by: |
| Oracles: | The validation form returns a list of warnings and errors to report any contravening of the DTD or bad design practice. If the document validates to the schema it outputs: ‘Document Validates OK’ |
Required Tools: The technique requires the following tools:
- Mozilla FireFox Web Browser
- Scholarly Technology Group’s XML Validation webpage
- Language Technology Group’s XML Validation webpage

Success Criteria: Both validators indicate that XTM files are well-formed and validated

Special Considerations:

5.1.4 Heuristic Evaluation

Table 4: Heuristic Evaluation Techniques

<table>
<thead>
<tr>
<th>Technique Objective:</th>
<th>Ensure all Topic Maps generated are in valid XTM syntax as defined by the XTM DTD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique:</td>
<td>Generated Topic Maps are tested on the XML validation form by the Scholarly Technology Group at brown university.</td>
</tr>
<tr>
<td>Oracles:</td>
<td>The validation form returns a list of warnings and errors to report any contravening of the DTD or bad design practice. If the document validates to the schema it outputs: ‘Document Validates OK’</td>
</tr>
<tr>
<td>Required Tools:</td>
<td>The technique requires the following tools:</td>
</tr>
<tr>
<td></td>
<td>- Mozilla FireFox Web Browser</td>
</tr>
<tr>
<td></td>
<td>- Scholarly Technology Group’s XML Validation webpage</td>
</tr>
<tr>
<td>Success Criteria:</td>
<td>Validator outputs: ‘Document Validates OK’</td>
</tr>
<tr>
<td>Special Considerations:</td>
<td></td>
</tr>
</tbody>
</table>

5.1.5 User Testing

Table 5: User Testing Techniques

| Technique Objective: | - To discover if user has access to functionality of project.                  |
|                      | - To discover if user has preference for a task, in comparable tools.           |
| Technique:           | Users perform tasks to achieve a certain goal, on our system and competitive SE tools. |
| Oracles:             | - User feedback on task, and tool preference.                                  |
|                      | - Users can also be timed for tasks.                                           |
| Required Tools:      | Rational Rose, Rational Requisite Pro, Developed System                        |
| Success Criteria:    | - All functions are performed by all users.                                    |
|                      | - Feedback of preference to Developed System.                                  |
### 5.1.6 Traceability of Test Cases

Figure 32 indicates an overview of the traceability between stakeholders’ requirements, use-case diagrams, implemented system modules and test cases.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Use-cases</th>
<th>Implementation</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open a Project</td>
<td>Create a Project</td>
<td>Add/Edit Requirements</td>
<td>Add/Edit Traceability</td>
</tr>
<tr>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
</tr>
<tr>
<td>Open Project</td>
<td>Create Project</td>
<td>Manage Requirements</td>
<td>Manage Traceability</td>
</tr>
<tr>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
</tr>
<tr>
<td>Open File</td>
<td>Create File</td>
<td>Requirements Manager</td>
<td>Traceability Manager</td>
</tr>
<tr>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
<td>Trace to</td>
</tr>
<tr>
<td>Usability Test Input</td>
<td>Usability Test Output</td>
<td>Usability Test Output</td>
<td>Usability</td>
</tr>
</tbody>
</table>

Each package will need to be tested by two test cases. The usability test case will require that the stakeholders’ functional requirements have been met by testing against use-case diagrams from Chapter 3. The test cases for tests of input and outputs will be tested by using XTM Validation Techniques described earlier.
5.2 Test Cases

5.2.1 Functional Testing

Testing of the final systems core functionality as specified by stakeholders’ requirements and use-case diagrams were tested to verify the systems compliance to the requirements specification.

5.2.1.1 Iterative Developer Testing

Since the Rational Unified Process was implemented for system development, testing of functionality was conducted iteratively at the end of each phase in which development took place. These tests ensured system compliance at the end of each phase of development.

5.2.1.2 User Testing

Since the goal of this thesis was not centred on usability of the final system, testing and validation metrics were not collected for user tests of this section. User testing was, however, conducted iteratively to ensure system compliance at the end of each development stage. User testing was also conducted after the final implementation of the system in order to ensure compliance with the stakeholders’ requirements.

5.2.1.3 Heuristic Evaluation

Heuristic evaluation was conducted by demonstrating implemented functionality and design to the stakeholder at the end of each phase in which development took place. The stakeholder was asked for feedback and any changes to the system requirements.

5.2.2 Validation of Outputs

Validation of XML Topic Map files generated during requirement management, traceability management and project creation phases were conducted in order to ensure the validity of the generated topic map structure.

The output files were validated with the official XTM Topic Maps Document Type Definition (DTD) which can be found at: http://www.topicmaps.org/xtm/1.0/xtm1.dtd
Two online validators were used in order to assure the validity of the topic map structures created. These validators may be found at:

- [http://www.stg.brown.edu/service/xmlvalid/](http://www.stg.brown.edu/service/xmlvalid/)
- [http://www.hcrc.ed.ac.uk/~richard/xml-check.htm](http://www.hcrc.ed.ac.uk/~richard/xml-check.htm)

### 5.2.3 Proof of Concept Testing

Validation of the concepts central to this thesis was accomplished by two tests in order to ensure validity of the research question of this thesis.

#### 5.2.3.1 Heuristic Evaluation

Heuristic evaluation was performed in order to ensure validation of the concept in the requirement traceability domain. The expert was shown the system and asked to for feedback regarding the visualisation of traceability as opposed to common practice devices such as traceability matrices and UML diagrams.

#### 5.2.3.2 User Testing

User testing was done in order to verify intuitiveness of the visual display, verification of the topic maps representation of UML models and navigability of the topic map structure. Users were shown examples of requirements traceability modelled in UML, traceability matrices as well as XML Topic Maps. Users were further asked to navigate the traceability structures by following certain traceability links. Feedback was received from each user and documented.
Chapter 6: Results and Findings

This chapter provides an overview of the results from implemented tests as well as further findings and insights gained from this thesis.

6.1 Results of Testing

6.1.1 Functional Testing

Iterative developer testing and user feedback throughout all phases of development were conducted in order to ensure system compliance to the stakeholders’ requirements. These tests were successful and allowed for continuous improvement of the systems functionality as required by the stakeholders’ requirements specification.

Final testing was completed using heuristic evaluation and user testing. The feedback was positive regarding the systems conformance to the functional requirements.

The system was therefore deemed to have adequately implemented all requirements as specified by the stakeholder.

6.1.2 Validation of Outputs

Continuous validation of topic map files created during requirements management, traceability management and project creation was conducted in order to ensure validity of files generated.

As per the validation techniques discussed in Chapter 5, these files were found to be valid as specified by the XML Topic Map Document Type Definition (DTD).

6.1.3 Proof of Concept Testing

Proof of concept testing was conducted using heuristic and user testing in order to verify the hypothesis presented in Chapter 1. Evaluators and users were tested using three separate concepts for the visual display of requirements traceability:

- Traceability matrix from Rational Requisite Pro
- UML diagrams modelling requirements traceability
- XML Topic Map visualisation of requirements traceability using the developed system
6.1.3.1 Heuristic Evaluation

Heuristic evaluation yielded positive feedback with the expert favouring the concept of visualising requirements traceability using XML Topic Maps. The expert generally viewed the visual overview of a requirement and its parents and dependencies as providing a more intuitive conceptual representation than competing techniques.

6.1.3.2 User Testing

User testing was conducted by showing users the three competing concepts in no particular order in order to minimise the possibilities of knowledge acquisition assisting in the understanding of concepts.

Users were quickly able to grasp all three concepts and indicated a strong favouritism towards representations using UML and XML Topic Maps over that of traceability matrices.

Although user’s indicated a preference for UML in terms of navigability, they generally preferred the concept of requirements traceability modelling using XML Topic Maps for a concise overview of a specific topic and its parents and dependencies.
Chapter 7: Conclusions

In this chapter, an overview of the most notable conclusions is summarised for the reader. Conclusions present the insights gained throughout this thesis along with the summaries of associated problems.

7.1 The Hypothesis is Probably True

The hypothesis from Chapter 1 was concluded to probably be true due to heuristic and user evaluations conducted. The visualisation of requirements traceability provided by XML Topic Maps was seen to contain several benefits for the intuitive display of requirements, their attributes and their relationships.

Navigability of the topic map structure did, however, meet with some criticism due to a lack of overall intuitiveness and ease of use. Users tended to prefer the general overview provided by UML, despite its unmanageability given large complexities.

7.2 The TMNav Visualisation of Topic Maps is a Problem

The touchgraph visualisation provided by TMNav was concluded to be a major factor in navigation problems associated with XML Topic Maps.

User’s indicated a strong preference for a general overview of all requirements in the data structure. Such an overview is not provided by the TMNav visualisation techniques.

An improved method of visualisation indicating an overview of all requirements and their relationships would be strongly preferred by users.

7.3 Requirements Traceability can be Effectively Managed using XML Topic Maps

The management of requirements traceability using the developed system was successfully implemented. The ease of implementation and subsequent use gives an indication of the effectiveness of requirements traceability management using XML Topic Maps.
The Functionality of the System is Adequate

Functional testing conducted indicated that stakeholders’ requirements were successfully met.

The implemented system allowed for all necessary functions as described by the stakeholder and the implementation of the final system is therefore seen as being successful.
Chapter 8: Lessons Learnt and Future Work

This chapter gives a brief overview of some of the key lessons learnt throughout this thesis. Future work on both this thesis and the core concept central to it is presented.

8.1 Lessons Learnt

Lessons learnt during the research and implementations of this thesis are described below. These lessons contribute to software engineering and requirements traceability by providing a brief overview of successes and difficulties met throughout the project.

8.1.1 Topic Maps provide an intuitive visualisation of requirements traceability

The visualisation of requirements traceability as provided by XML Topic Maps is seen as intuitive and provides an improvement over existing techniques. This technique for the visualisation of requirements traceability provides an intuitive overview of a requirement, its relationships and attributes.

8.1.2 A generalised overview of all requirements and their relationships is required

Navigability of XML Topic Map structures for visualisation of requirements traceability requires a more generalised overview of all requirements and their relationships. This generalised overview should provide the user with a considerably enhanced overview of the traceability existing between requirements as well as allow for the intuitive navigation through requirements of interest.

8.2 Future Work

This section describes directions of future work that may be applied to both this thesis and the concept of requirements traceability visualisation using XML Topic Maps. This section provides a guideline for the possible directions of research that may be undertaken on this topic in future.

5.2.4 Future work on this thesis

Extensions to this thesis may follow by following the suggestions below:
5.2.4.1 Improvement of topic map visualisation techniques

The visualisation component of the implemented system should be improved in order to allow for a general overview of the entire requirements traceability system. This would allow for the intuitive navigation required by users as documented under conclusions.

5.2.4.2 Integration with existing case tools

The system should be extended to integrate with existing case tools. This would allow for the implementation and use of various software engineering methods as well as the seamless import and export of data between software packages. Additionally, the system should be extended to allow for the definition of use-cases linking to functional requirements.

5.2.5 Future work in related research

The concept described throughout this thesis has the potential to be extended on in a various number of ways. A few of these are discussed below:

5.2.5.1 Constraining Topic Maps

The Topic Map Constraint Language should be research in order to allow for constraints to be placed on Topic Maps which represent requirements traceability. The constraints could be created to allow for the implementation of patterns – best practice solutions for the implementation of requirements traceability.

5.2.5.2 Use of design patterns

Design patterns could be used to create constraints on topic map structures. These constraints would represent best practice design for the implementation of traceability patterns.

5.2.5.3 Defining XML Schemas for Topic Maps

Best practice solutions for requirements traceability could further be extended by the use of XML Schemas for Topic Maps. Schemas provide an improved form of constraint and may be research as a potential constraint language for XML Topic Maps.
5.2.5.4 Change Impact Analysis

Change impact analysis should be research and implemented in topic maps for the automation of change throughout all requirements. A possible technique of calculating the automation of impact analysis could be Bayesian Networks which should be researched in depth.
Chapter 9: References


International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE ’2003), Montreal, Canada, 10-2003.


in Emerging Forms of Software Engineering (TEFSE ’2003), Montreal, Canada, 10-2003.


Appendix A – Rational Unified Process Phases
(from Iteration Plan)

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Management</td>
</tr>
<tr>
<td>2</td>
<td>Conceive New Project</td>
</tr>
<tr>
<td>5</td>
<td>Evaluate Project Scope and Risk</td>
</tr>
<tr>
<td>7</td>
<td>Develop Software Development Plan</td>
</tr>
<tr>
<td>15</td>
<td>Plan Remainder of Initial Iteration</td>
</tr>
<tr>
<td>17</td>
<td>Manage Iteration</td>
</tr>
<tr>
<td>20</td>
<td>Monitor and Control Project</td>
</tr>
<tr>
<td>25</td>
<td>Reevaluate Project Scope and Risk</td>
</tr>
<tr>
<td>27</td>
<td>Plan for Next Iteration</td>
</tr>
<tr>
<td>29</td>
<td>Refine Software Development Plan</td>
</tr>
<tr>
<td>37</td>
<td>Requirements</td>
</tr>
<tr>
<td>38</td>
<td>Analyze the Problem</td>
</tr>
<tr>
<td>42</td>
<td>Understand Stakeholder Needs</td>
</tr>
<tr>
<td>47</td>
<td>Define the System</td>
</tr>
<tr>
<td>50</td>
<td>Manage the Scope of the System</td>
</tr>
<tr>
<td>53</td>
<td>Refine the System Definition</td>
</tr>
<tr>
<td>56</td>
<td>Manage Changing Requirements</td>
</tr>
<tr>
<td>61</td>
<td>Test</td>
</tr>
<tr>
<td>62</td>
<td>Define Evaluation Mission</td>
</tr>
<tr>
<td>63</td>
<td>Identify Test Motivations</td>
</tr>
<tr>
<td>64</td>
<td>Agree Mission</td>
</tr>
<tr>
<td>65</td>
<td>Identify Targets of Test</td>
</tr>
<tr>
<td>66</td>
<td>Define Assessment and Traceability Needs</td>
</tr>
<tr>
<td>67</td>
<td>Identify Test Ideas</td>
</tr>
<tr>
<td>68</td>
<td>Define Test Approach</td>
</tr>
</tbody>
</table>

Inception Plan

![Inception Plan Gantt Chart](chart.png)
Elaboration Plan

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Management</td>
<td>7 days</td>
</tr>
<tr>
<td>2</td>
<td>Manage Iteration</td>
<td>5 days</td>
</tr>
<tr>
<td>6</td>
<td>Monitor and Control Project</td>
<td>7 days</td>
</tr>
<tr>
<td>10</td>
<td>Reevaluate Project Scope and Risk</td>
<td>1 day</td>
</tr>
<tr>
<td>12</td>
<td>Plan for Next Iteration</td>
<td>1 day</td>
</tr>
<tr>
<td>15</td>
<td>Refine Software Development Plan</td>
<td>1 day</td>
</tr>
<tr>
<td>25</td>
<td>Requirements</td>
<td>7 days</td>
</tr>
<tr>
<td>26</td>
<td>Analyze the Problem</td>
<td>2 days</td>
</tr>
<tr>
<td>30</td>
<td>Understand Stakeholder Needs</td>
<td>2 days</td>
</tr>
<tr>
<td>35</td>
<td>Define the System</td>
<td>2 days</td>
</tr>
<tr>
<td>33</td>
<td>Manage the Scope of the System</td>
<td>2 days</td>
</tr>
<tr>
<td>41</td>
<td>Refine the System Definition</td>
<td>4 days</td>
</tr>
<tr>
<td>45</td>
<td>Manage Changing Requirements</td>
<td>4 days</td>
</tr>
<tr>
<td>43</td>
<td>Analysis and Design</td>
<td>7 days</td>
</tr>
<tr>
<td>50</td>
<td>Define a Candidate Architecture</td>
<td>1 day</td>
</tr>
<tr>
<td>53</td>
<td>Analyze Behavior</td>
<td>3 days</td>
</tr>
<tr>
<td>57</td>
<td>Design Components</td>
<td>5 days</td>
</tr>
<tr>
<td>61</td>
<td>Refine the Architecture</td>
<td>5 days</td>
</tr>
<tr>
<td>65</td>
<td>Implementation</td>
<td>6 days</td>
</tr>
<tr>
<td>63</td>
<td>Structure the Implementation Model</td>
<td>2 days</td>
</tr>
<tr>
<td>71</td>
<td>Plan the Integration</td>
<td>1 day</td>
</tr>
<tr>
<td>73</td>
<td>Implement Components</td>
<td>4 days</td>
</tr>
<tr>
<td>79</td>
<td>Integrate Each Subsystem</td>
<td>1 day</td>
</tr>
<tr>
<td>81</td>
<td>Integrate the System</td>
<td>1 day</td>
</tr>
<tr>
<td>83</td>
<td>Test</td>
<td>7 days</td>
</tr>
<tr>
<td>84</td>
<td>Define Evaluation Mission</td>
<td>1 day</td>
</tr>
<tr>
<td>91</td>
<td>Verify Test Approach</td>
<td>1 day</td>
</tr>
<tr>
<td>97</td>
<td>Test and Evaluate</td>
<td>2 days</td>
</tr>
</tbody>
</table>
**Construction Plan**

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Management</td>
<td>5 days</td>
</tr>
<tr>
<td>2</td>
<td>Manage Iteration</td>
<td>3 days</td>
</tr>
<tr>
<td>6</td>
<td>Monitor and Control Project</td>
<td>4 days</td>
</tr>
<tr>
<td>10</td>
<td>Reevaluate Project Scope and</td>
<td>1 day</td>
</tr>
<tr>
<td>12</td>
<td>Plan for Next Iteration</td>
<td>1 day</td>
</tr>
<tr>
<td>15</td>
<td>Refine Software Development</td>
<td>1 day</td>
</tr>
<tr>
<td>25</td>
<td>Requirements</td>
<td>1 days</td>
</tr>
<tr>
<td>26</td>
<td>Manage Changing Requirements</td>
<td>1 days</td>
</tr>
<tr>
<td>29</td>
<td>Analysis and Design</td>
<td>2 days</td>
</tr>
<tr>
<td>30</td>
<td>Design Components</td>
<td>0.5 days</td>
</tr>
<tr>
<td>34</td>
<td>Refine the Architecture</td>
<td>2 days</td>
</tr>
<tr>
<td>41</td>
<td>Implementation</td>
<td>5 days</td>
</tr>
<tr>
<td>42</td>
<td>Plan the Integration</td>
<td>1 day</td>
</tr>
<tr>
<td>44</td>
<td>Implement Components</td>
<td>1 days</td>
</tr>
<tr>
<td>49</td>
<td>Integrate the System</td>
<td>1 day</td>
</tr>
<tr>
<td>51</td>
<td>Test</td>
<td>5 days</td>
</tr>
<tr>
<td>52</td>
<td>Define Evaluation Mission</td>
<td>1 day</td>
</tr>
<tr>
<td>59</td>
<td>Verify Test Approach</td>
<td>1 day</td>
</tr>
<tr>
<td>65</td>
<td>Test and Evaluate</td>
<td>1 day</td>
</tr>
</tbody>
</table>

**Transition Plan**

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Management</td>
<td>4 days</td>
</tr>
<tr>
<td>2</td>
<td>Manage Iteration</td>
<td>3 days</td>
</tr>
<tr>
<td>6</td>
<td>Monitor and Control Project</td>
<td>4 days</td>
</tr>
<tr>
<td>10</td>
<td>Close-Out Project</td>
<td>1 day</td>
</tr>
<tr>
<td>13</td>
<td>Requirements</td>
<td>4 days</td>
</tr>
<tr>
<td>14</td>
<td>Manage Changing Requirements</td>
<td>4 days</td>
</tr>
<tr>
<td>17</td>
<td>Analysis and Design</td>
<td>3 days</td>
</tr>
<tr>
<td>18</td>
<td>Refine the Architecture</td>
<td>3 days</td>
</tr>
<tr>
<td>25</td>
<td>Implementation</td>
<td>4 days</td>
</tr>
<tr>
<td>26</td>
<td>Plan the Integration</td>
<td>1 day</td>
</tr>
<tr>
<td>28</td>
<td>Implement Components</td>
<td>2 days</td>
</tr>
<tr>
<td>34</td>
<td>Integrate Each Subsystem</td>
<td>2 days</td>
</tr>
<tr>
<td>36</td>
<td>Integrate the System</td>
<td>2 days</td>
</tr>
<tr>
<td>38</td>
<td>Test</td>
<td>4 days</td>
</tr>
<tr>
<td>39</td>
<td>Define Evaluation Mission</td>
<td>1 day</td>
</tr>
<tr>
<td>40</td>
<td>Test and Evaluate</td>
<td>3 days</td>
</tr>
</tbody>
</table>