2015

Use of IBM Collaborative Lifecycle Management Solution to Demonstrate Traceability for Small, Real-World Software Development Project

Lovelesh Chawla
University of North Florida, n00850047@ospreys.unf.edu

USE OF IBM COLLABORATIVE LIFECYCLE MANAGEMENT SOLUTION TO DEMONSTRATE TRACEABILITY FOR SMALL, REAL-WORLD SOFTWARE DEVELOPMENT PROJECT

by

Lovelesh Chawla

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computing and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

November, 2015

The thesis "Use of IBM Collaborative Lifecycle Management Solution to Demonstrate Traceability for Small, Real-World Software Development Project" submitted by Lovelesh Chawla in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee: Date

_____ _____
Dr. Robert F. Roggio
Thesis Advisor and Committee Chairperson

_____ _____
Dr. Lakshmi Goel

_____ _____
Dr. Sherif A. Elfayoumy

Accepted for the School of Computing:

_____ _____
Dr. Sherif A. Elfayoumy
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

_____ _____
Dr. Mark A. Tumeo
Dean of the College

Accepted for the University:

_____ _____
Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENT

I would like to express my special appreciation to my advisor, Dr. Robert F. Roggio for being a tremendous mentor for me. I would like to thank him for all his advice and inputs throughout the realization of my thesis. His advice on this thesis and the suggestions to improve materials presented were of great help. I would like to thank Dr. Sherif A. Elfayoumy and Dr. Lakshmi Goel, for serving as my committee members even through their busy schedules and for their valuable suggestions and advice that were tremendously helpful to complete this thesis.

I am extremely grateful to Larry Snedden for installing and maintaining Rational Team Concert, Rational Requirement Composer and Rational Quality Manager on School of Computing server.

A special thanks to Mr. Jim Littleton for reviewing my thesis report and ensuring it is in compliance with thesis document requirements. I am also thankful to Dr. Roger Eggen for navigating me through the Graduate School and School of Computing processes.

Finally, I want to thank my wife Kelly for her patience, and support at all times.

CONTENTS

FIGURES

TABLES

ABSTRACT

The Standish Group Study of 1994 showed that 53 percent of software projects failed outright and another 31 percent were challenged by extreme budget and/or time overrun. Since then different responses to the high rate of software project failures have been proposed. SEI's CMMI, the ISO's 9001:2000 for software development, and the IEEE's JSTD-016 are some examples of such responses. Traceability is the one common feature that these software development standards impose.

Over the last decade, software and system engineering communities have been researching subjects such as developing more sophisticated tooling, applying information retrieval techniques capable of semi-automating the trace creation and maintenance process, developing new trace query languages and visualization techniques that use trace links, applying traceability in specific domains such as Model Driven Development, product line systems and agile project environment. These efforts have not been in vain. The 2012 CHAOS results show an increase in project success rate of 39% (delivered on time, on budget, with required features and functions), and a decrease of 18% in the number of failures (cancelled prior to completion or delivered and never used). Since research has shown traceability can improve a project's success rate, the main purpose of this thesis is to demonstrate traceability for a small, real-world software development project using IBM Collaborative Lifecycle Management.

The objective of this research was fulfilled since the case study of traceability was described in detail as applied to the design and development of the Value Adjustment Board Project (VAB) of City of Jacksonville using the scrum development approach within the IBM Rational Collaborative Lifecycle Management Solution. The results may benefit researchers and practitioners who are looking for evidence to use the IBM CLM solution to trace artifacts in a small project.

Chapter 1

INTRODUCTION

1.1 Traceability

Traceability is defined as “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match” [IEEE90].

This definition focuses on establishing a relationship between two elements that leads to a third element called “Traceability Relationship” [Leffingwell02]. "Is fulfilled by," "is part of" and "is derived from," are some examples of traceability relationships. These relationships may differ depending on the various elements that are linked (for instance, a specific "validated by" relationship between X and Y establishes that Y is used to validate X) [Leffingwell02].

In the context of this paper, the traceability relationship can be simply understood as a basic "traced- to" and "traced-from" model (refer to Figure 1).

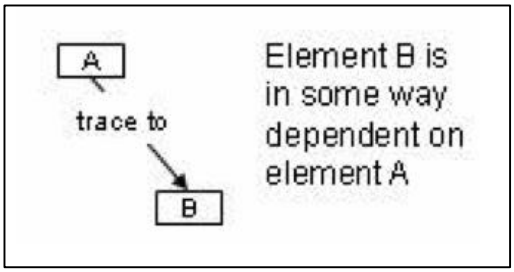


Figure 1. Simple Dependency Relationship [Leffingwell02]

1.2 Generalized Traceability Model

Distinct projects inherently result in various kinds of requirements artifacts and in the organizational structure of those artifacts. However, according to Leffingwell, there is a particular model for traceability strategy that is essentially static and very widely used. The strategy is 'hierarchical' in that it begins with first stating the higher-level requirements, followed by detailed requirements, implementation, and lastly testing of the requirements (refer to Figure 2) [Leffingwell02].

The 'static' model described about reveals that requirements are traced within the requirements domain, then further through the domains of implementation and testing. According to Leffingwell, some extraneous artifacts produced in development activities that can be traced are not shown in the figure. However, the basic types of traces that are shown in the figure usually cover most needs [Leffingwell02]. The following section covers examples based on this model and explains the importance of tracing using it.

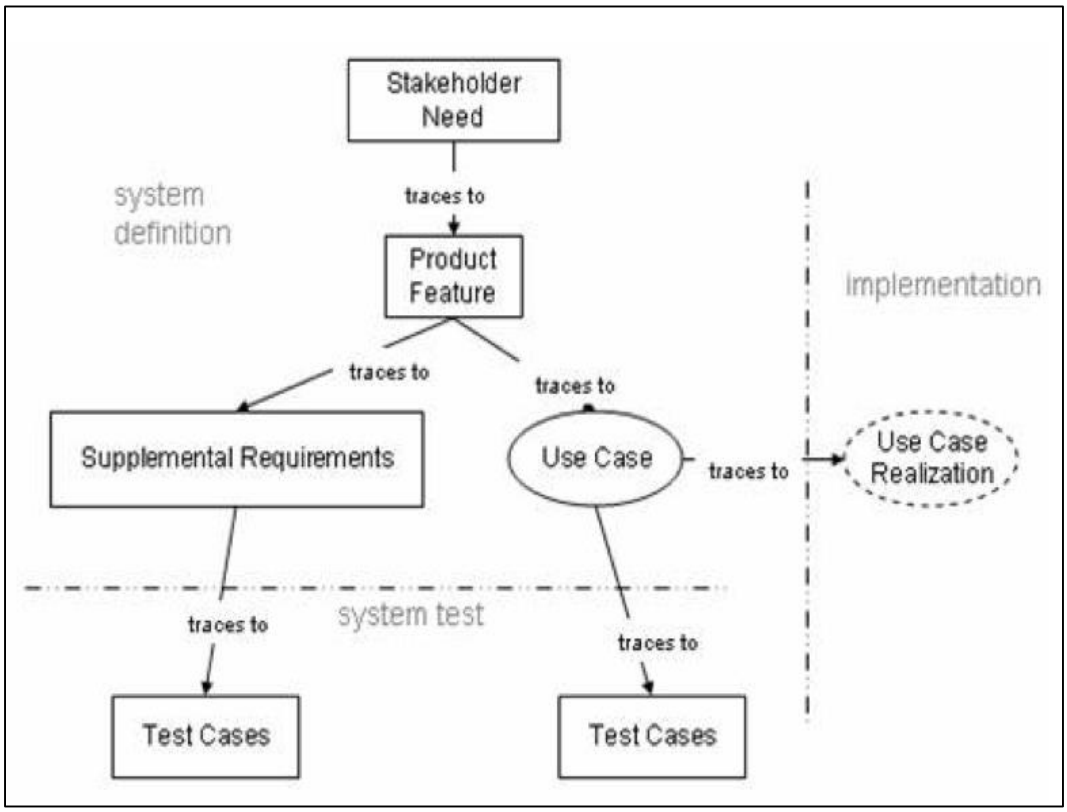


Figure 2. Generalized Traceability Hierarchy [Leffingwell02]

1.3 Examples of Traceability

1.3.1 Tracing Requirements in the System Definition Domain

Traces within this domain address traceability within the system or product definition domain. It is also called “requirement to requirement” traceability because it links a certain kind of requirement to another. For example, a feature requirement can be related to a use case requirement [Leffingwell02].

1.3.2 Tracing User Needs to Product Features

A system is usually developed to meet the needs of users and stakeholders, so it is crucial to understand those needs and define features that meet those needs. This can be achieved by continually relating the product features back to user needs. A simple table, or traceability matrix, can be used to relate them (refer to Table 1).

	Feature 1	Feature 2	….	Feature n
Need # 1	X			
Need # 2		X		X
Need …		X	X	
Need # m				X

Table 1. Traceability Matrix – User Needs to System Features [Leffingwell02]

"X" in the cell(s) indicates that an individual feature has been defined to support one or more user needs. Since these needs are specified at the higher level of abstractions there are usually far fewer needs identified than the number of system features that are defined to implement those needs. Hence, this is usually a "one to many" mapping,

A traceability matrix can be examined for possible evidence of error such as:

1. If a row contains no X's whatsoever, there is a chance that no feature has been defined to meet a user need. Noting such a pattern is a clear indication that it should be re-analyzed, and a feature may need to be defined to meet the need.

2. Observing a lack of X’s in a column may indicate that an extraneous feature exists which matches no particular user need, that the purpose of the feature may have been misidentified, or that a particular feature is no longer used but remains in the system unnecessarily.

Modern requirements management tools are able to automatically inspect the possible conditions of error named above. [Leffingwell02]. Once the need-feature relationship is mapped and confirmation is made that user needs and features are defined, understood and met, the next stage of the hierarchical traceability model begins. Here, relationships between features and use cases are identified.

1.3.3 Tracing Features to Use Cases

According to Leffingwell, tracing system features to the system's use cases and tracing user needs to system features are of equal significance, because the use case reveals the user’s perspective to implement a system [Leffingwell02].

As before, a simple matrix can be used to relate them for all systems; however, using matrix can be tedious and complex for large systems. The system features are listed in the row headers and use cases are listed in the column headers (refer to Table 2). Once the rows (features) and columns (use cases) are defined, an (X) is marked in in the cell(s) to represent the traceability. This is usually a set of "many to many" relationships. One or

multiple use case may be created to implement a feature. But it is also possible that one use case implements multiple features.

	Use case 1	Use case 2	….	Use case n
Feature # 1	X			X
Feature # 2		X		X
Feature …			X	
Feature # m		X		X

Table 2. System Features versus Use Cases [Leffingwell02]

After all the known feature-to-use-case relationships are established the traceability matrix can be examined for potential indications of error:

1. If a *row* does not have any Xs, it means a use case may be defined to include the feature.
2. If a *column* does not have any Xs, it means a use case may not be required and does not include any desired features.

Once the feature-use-case relationships are mapped and it is verified that the features and use cases are correctly developed, a similar concept can be applied to the non-functional requirements and their specification.

1.3.4 Tracing Features to Supplementary Requirements

"Even though use cases include the majority of the definition of the system's functional behavior, supplementary requirements also hold valuable system behavioral requirements" [Leffingwell02]. These generally include the non-functional requirements of the system such as usability, reliability, supportability, and so on. The vision document may often contain these requirements or additional high-level requirements. This can be shown using similar matrix (refer to Table 3).

	Supplementary Req 1	Supplementary Req 2	….	Supplementary Req n
Feature or System Requirement #1	X			X
Feature or System Requirement #2		X		X
Feature or System Requirement #3		X		X

Table 3. System Features versus Supplementary Requirements [Leffingwell02]

1.3.5 Tracing Requirements to Implementation

Following the tracing requirements in the system definition domain (requirement-to-requirement traceability), are the implementation and test domains. The following section covers different traceability models of requirements to implementation [Leffingwell02].

1.3.6 Tracing Use Cases to Use-Case Realizations

In this model use case (artifact of definition domain) is traced to use-case realization (artifact of implementation domain) to map design and requirements (refer to Figure 3).

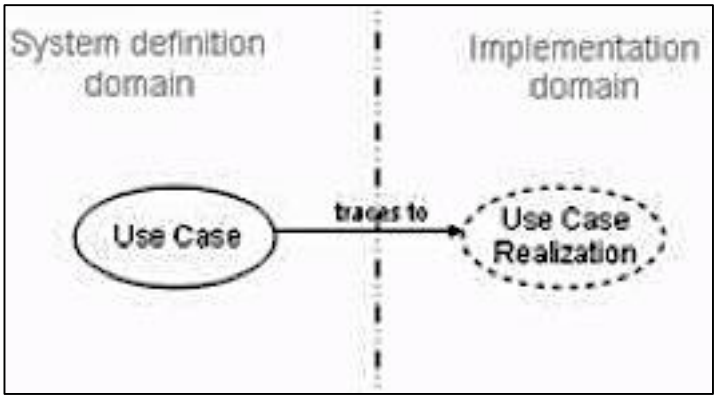


Figure 3. Tracing from Requirements to Implementation [Leffingwell02]

There is a one-to-one relation between a use case and its realization that simplifies the traceability problem immensely and does not require a matrix for the analysis [Leffingwell02].

1.3.7 Tracing from the Use-Case Realization into Implementation

Traceability to code can be established after the development of the use-case realization [Leffingwell02]. In this traceability relationship to implement the collaboration, a use-case realization is followed to its components such as the classes/code, subsystems, packages, etc. (refer to Figure 4).

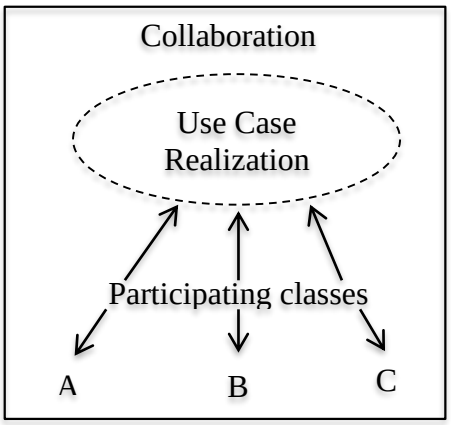


Figure 4. Tracing from the Use-Case Realization to Classes [Leffingwell02]

Types of tools used in requirements, analysis, and design efforts are important factors in achieving this traceability. In absence of the adequate tools this tractability can become very difficult and can lead to hundreds of use cases and thousands of classes.

1.3.8 Tracing Supplementary Requirements into Implementation

Some users use user stories, lists, and other formats rather than use-case form to express the requirements. It is important to trace non-functional requirements from

supplementary requirements into implementation to accomplish high traceability. In this case, a single requirement, or a set of requirements are traced to collaborate in the implementation. The collaboration is named and the tool tracks the links using some special ways (refer to Figure 5). Specific code in classes can be traced from here that realizes the collaboration. The mechanics of this can differ depending on the type of tool used.

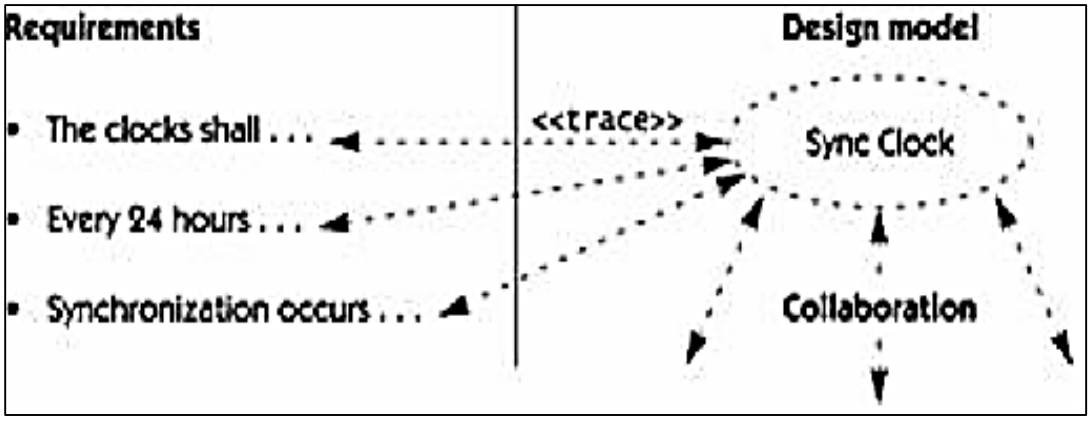


Figure 5. Tracing from Supplementary Requirements to Implementation [Leffingwell02]

1.3.9 Tracing from Requirements to Test

This is the last system domain used to trace requirements from the requirement domain to the testing domain.

1.3.10 Tracing from Use Case to Test Case

"One specific approach to comprehensive testing is to ensure that every use case is 'tested by' one or more test cases" [Heumann01]. This can be seen in the generalized traceability hierarchy (refer to Figure 6).

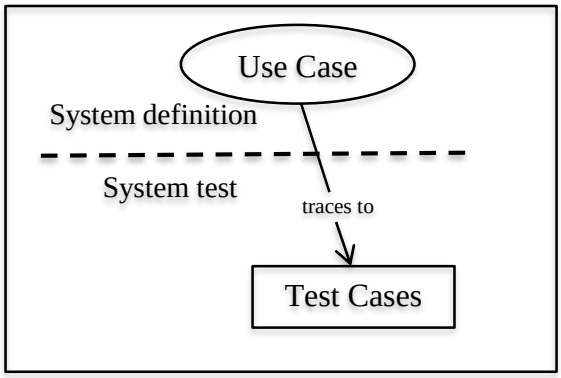


Figure 6. Tracing Use Cases to Test Cases [Leffingwell02]

Usually this relationship is not a 1-to-1 transition as shown in the figure. It is a one-to-many relationship, as an elaborated use case will have many scenarios to be tested for. All these scenarios should trace back to their specific use case (refer to Figure 7).

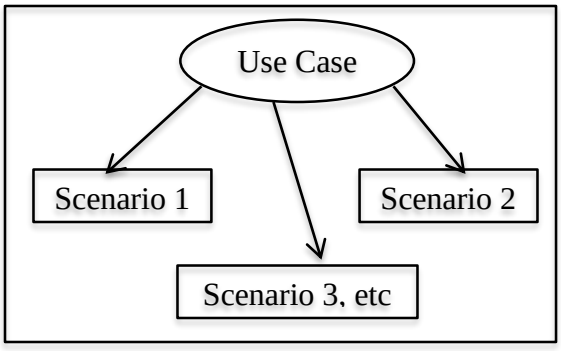


Figure 7. Tracing Use Cases to Test Case Scenarios [Leffingwell02]

This traceability can also be described using a matrix. Each row is a scenario of a particular use case (refer to Table 4). However, each scenario may require multiple test cases (refer to Figure 8). The can be shown in the matrix form by adding one more column to the matrix. So relationship among these artifacts can be shown by using a couple one-to-many matrix (use case to scenario and scenario to test case)

Use Case	Scenario Number	Originating Flow	Alternate Flow	Next Alternate	Next Alternate
Use Case Name #1	1	Basic Flow			
	2	Basic Flow	Alternate Flow 1		
	3	Basic Flow	Alternate Flow 1	Alternate Flow 2	
	4	Basic Flow	Alternate Flow 3		
	5	Basic Flow	Alternate Flow 3	Alternate Flow 1	
	6	Basic Flow	Alternate Flow 3	Alternate Flow 1	Alternate Flow 2
	7	Basic Flow	Alternate Flow 4		
	8	Basic Flow	Alternate Flow 3	Alternate Flow 4	
Use Case Name #2	1	Basic Flow			

Table 4. Traceability Matrix for Use Case to Scenarios [Leffingwell02]

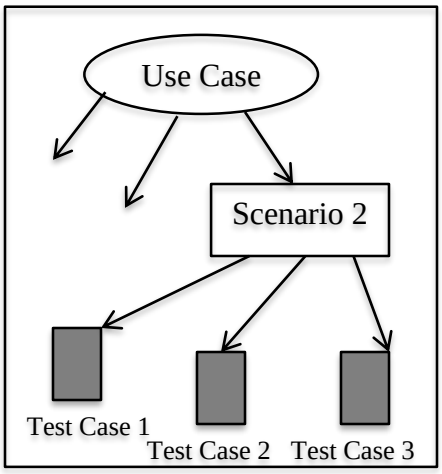


Figure 8. Tracing Scenarios to Test Cases [Leffingwell02]

Use Case	Scenario Number	……	Test Case Id
Use Case # 1	1		1.1
	2		2.1
	3		3.1
	4		4.1
	4		4.2
	4		4.3
	5		5.1
	6		6.1
	7		7.1
	7		7.2
	8		8.1
Use Case #2	1		1.1

Table 5. Traceability Matrix for Use Cases to Test Cases [Leffingwell02]

1.3.11 Tracing from Supplementary Requirements to Test Cases

The process to trace non-functional requirements to test cases is like the requirements-to-implementation process explained above. In this traceability relationship, each requirement is either separately traced to a scenario and a test case, or requirements are grouped into "requirements packages" that uses the logic similar to a use case. The above matrix can be used to show this relationship but the far most left column will change to “specific requirement” or “requirements package”.

1.4 Importance of Traceability

Properly tracing requirements artifacts through initiation, design, implementation, and testing phases significantly helps to implement successful software. According to Marco Leon there are several advantages of tracing requirements, some of which are:

- An easier coverage analysis: All requirements can be traced back to higher level requirements. Therefore, it can be validated that all requirements are fulfilled.
- A Better Design: Requirement tracing provides complete information to the architect.
- Fewer Code Reworks: Through tracing, possible issues can be discovered and addressed at an earlier stage.
- Improved Change Management: When a requirement is changed, the entire “trace” can be reviewed for the potential impact to the application.

All of these benefits reduce the software development cycle [Leon00].

Ecklund, Delcambre and Freiling focus on the importance of traceability in change management. They state that “Traceability allows us to gauge the scope of a change with respect to any level of a system’s evolving design by following the traceability links forward from affected use cases to the level we are examining. Understanding the scope of a change on any level enables us to make judicious decisions about how to change the design at that level” [Ecklund96].

Domges and Paul focus on benefits of traceability in maintenance of a system. They state that requirements traceability is essential to maintain a system efficiently. If traceability is not implemented, it can drop the quality of the system. This may increase cost and time to complete the project. During that time if important team members quit the project, it can lead to loss of knowledge, which later can result in wrong decisions, misunderstandings, and miscommunication [Domges08].

Chapter 2

BACKGROUND

2.1 History

During the NATO Software Engineering Conference of 1968, the function of traceability in solving many fundamental problems of the industry was first addressed [Peter69]. Three projects (each spanning one year) targeted methodology and focused on ensuring that systems "contain explicit traces of the design process." Conference critics positively received these projects. Additionally, a particular paper focused on two issues: analyzing the requirements for an effective methodology of computer system design, and emphasizing the importance of developing systems to reflect their designs. [Randell68]

One of the pioneering surveys on future trends in software engineering focused heavily on traceability [Barry86]. Traceability was implemented in projects that would depend on tool support during early stages of development. In the 1980's, national and global standards for system development included the use of traceability. The DOD-STD-2167A is an example of one such system [Dorfman90]. In the late 1990s published research reflected a 'branching out' in the area of traceability.

Expanding research at the beginning of the new millennium led to new discoveries in model-driven development [Ismenia07] and automated traceability [Steven06]. In 1980s new commercial tools to support traceability were introduced, and in 1990s and 2000s significant development was observed in this area. However, in reality practice of traceability is still not well documented. [Mader09].

In the recent years, different methods have been developed by the software and system engineering societies to address distinct areas of traceability [Spanoudakis05].

Research into software traceability has been primarily focused on

- "The study and definition of different types of traceability relations" [Zisman03, Dick02, Egyed02, Gotel95, Kaindl92, Leteiler02, Lindval96, Pohl96, Ramesh92, Spanoudakis04]
- "Support for the generation of traceability relations" [Alexander02, Antoniol02, Egyed02, Pinheiro96, Maletine03, Marcus03, Spanoudakis04]
- "Development of architectures, tools, and environments for the representation and maintenance of traceability relations" [Cleland-Huang02, Cleland-Huang03, Sherba03, Pinheiro96]
- "Empirical investigations into organizational practices regarding the establishment and deployment of traceability relations in the software development life cycle" [Arkley, Binachi00, Gotel94, Ramesh98, Ramesh01, Strens96, Lindval96].

Due to difficulty in generating precise and accurate traceability relationships automatically, many modern projects do not implement effective traceability [Arkley, Ramesh01]. A majority of the existing tools either forces the user to identify the traceability relations manually [Huan02, Gotel95, Pohl96] or provide traceability generation methods that do not establish precise and accurate relations ([Alexander02, Antoniol02, Marcus03]). In the first case, the efforts to manually establish traceability relations outweigh the expected benefits of traceability for large projects. This reduces the possibilities of an organization to implement traceability, unless it is a regulatory mandate. Lack of accurate and precise relations leads to ambiguous traceability between artifacts and makes it of little use in the second case. Therefore, traceability methods are not extensively used in the industry.

The following sections in this chapter will show a roadmap of research and practices in software traceability by discussing:

- "Different frameworks and classifications of traceability relations"
- "Different approaches to the generation of traceability relations including manual, semi-automatic and automatic approaches"
- "Different approaches to the representation, recording, and maintenance of traceability relations that underpin the architectural design and implementation of traceability tools and environments,"
- "Different ways of deploying traceability relations in the software development process" [Spanoudakis05].

2.2 Different Frameworks and Classifications of Traceability Links

"Traceability links can usually be divided into horizontal traceability or vertical traceability" links [Lindval96]. The first links are between different models, and the second links are between artifacts of the same model. According to Klaus Pohl there is another way to categorize traceability relations into "18 different types of relations organized in five different groups" [Pohl96]. These groups are:

- "Condition Link Group" - This group contains links between requirements and their related restrictions.
- "Content Link Group" - This group contains links that show resemblance and differences between requirements.
- "Documentation Link Group" - This group contains links between different kinds of software documents and requirement.
- "Evolutionary Link Group" - This group contains replacement links between requirements ("e.g. a requirement X has replaced a requirement Y in requirements document").
- "Abstraction Link Group" - This group contains abstract relations such as generalization and refinement between requirements [Pohl96].

According to George Spanoudakis and Andrea Zisman various types of traceability links proposed earlier can primarily be organized into eight categories: "dependency, generalization/refinement, evolution, satisfaction, overlap, conflicting, rationalization, and contribution relations" [Spanoudakis05]. These categories are explained below. The

term artifact is commonly used in the explanation of these categories to show the different traceable entities, and objects in software. “Examples of these elements are stakeholders, requirements statements, design components (e.g. classes, states), code statements, test data, etc.” [Spanoudakis05]. In this classification two artifacts a1 and a2, may be linked by more than one type of link.

In a dependency relation, an artifact a1 depends on an artifact a2. If a1 changes these changes must be reflected in a2. Ramesh and Jarke state that dependency relations exist between various requirements, as well as between requirements and design elements. According to them, dependency relations show hierarchies of artifacts and dependency between them, and can be used to manage requirements [Ramesh01]. Von Knethen states that dependency relations are used between “documentation entities (e.g. textual requirements, use cases) and logical entities (e.g. function, tasks) to assist with fine-grained impact analysis” [Knethen02, Paech02]. In contrast, Spanoudakis and Zisman say “dependency relations are called *requires-feature-in relations* and associate parts of use case specifications and customer requirements specifications” [Spanoudakis04, Kraus03]. The requires-feature-in relations mean that without the structural and functional features of a requirement, specific part of a use case cannot be realized, “or that one requirement depends on the existence of a feature required by another requirement” [Spanoudakis04, Kraus03]. Maletic calls dependency relations as *causal conformances* that suggest related software documents are produced in an order (e.g., test cases can only be produced after use cases) [Maletic03]. However, Gotel and Finkelstein call dependency relations *developmental relations* that provide requirement traceability through the

development of artifacts that are created during the other phases of the software development life cycle [Gotel95]. Dependency relations have also been used for requirements [Alexander03, Pohl96], "scenarios, code, and model elements" [Egyed03], "and to support the design and implementation of product lines" [Riebisch01].

A generalization/refinement relation can recognize complex component of a system and break them down into simpler components, or combine components of a system to create other components, or refine a component by another component. Pohl claims "these relations are classified as *abstraction links* and represent abstractions between trace requirements" [Pohl96]. Xu and Ramesh say that "generalization/refinement relations are also used to support associations between business process, decision, and workflow system objects" [Xu02]. Gotel claims that they are called *containment relations*, as these relations combine requirement artifacts to form a composite artifact [Gotel95].

An evolution relation signifies the evolution of software artifacts. In this case an artifact a1 evolves into an artifact a2. This occurs because a1 is replaced by a2 during the development, maintenance, or evolution of the system. Pohl confirms that these types of relations are called *replace*, *formalize*, and *elaborate* relations and should be used to associate requirements [Pohl96]. In contrast, Pinheiro and Goguen call evolution relations as *replace* and *abandon* relations. A replace relation shows that a requirement is modified. An abandon relation shows that a requirement is unnecessary and is rejected [Pinheiro96]. Maletic says that "evolution relations are called *non-causal conformance* relations" and do not show a clear connection between artifacts [Maletic03]. Different

version of the same document is an example of this relation. Gotel and Finkelstein mention that evolution relations are called *temporal relations.* They show how an artifact has evolved during its development [Gotel95]. Constantopoulos states that evolution relations are used to "signify derivations between requirements, design, and code artifacts" [Constantopoulous95].

In a satisfiability relation, an artifact a1 satisfies an artifact a2. "This means a1 meets the expectation, needs, and desires of a2, or a1 complies with a condition represented by a2" [Spanoudakis05]. As per Jarke and Ramesh, satisfiability relations connect requirements and system components (e.g. design components) and guarantee that a system 'satisfies all the requirements [Jarke01].

An overlap relation is between two artifacts a1 and a2, "which refer to a common feature of the underlying system" [Spanoudakis05]. Spanoudakis conveys that "overlap relations are used between requirement statements, use cases, and analysis object models" [Spanoudakis04]. Pohl claims that an overlap relation links various documents such as test case, comment, background information, and examples with requirements [Pohl96]. Von Knethen states that "overlap relations are called *representation* relations and connect document that shows the same logical entity" [Knethen02]. Gotel mentions that "overlap relations are called *adopts relations* (a subtype of *connectivity* relation) and are used to associate requirement artifacts" [Gotel94].

A conflict relation "signifies conflicts between two artifacts a1 and a2 (e.g., when two requirements conflict with each other)" [Spanoudakis05]. As per Jarke and Ramesh, conflict relations are used to show conflicts between artifacts and to provide information that can be used to resolve the issues [Jarke01].

A rationalization relation is "used to represent and maintain the rationale behind the creation and evolution of artifacts" [Spanoudakis05]. Jarke and Ramesh use rationalization relations to capture the history of how artifacts are created [Jarke01]. Leteiler02 states, "rationalization relations are expressed between traceable specifications (a software specification with different level of granularity such as document, model, diagram, use case, etc.) and a rationale specification (a document containing assumptions or alternatives to a traceable specification)" [Leteiler02].

A contribution relation is used to show connection between requirement artifacts and its stakeholders. Gotel and Finkelstein proposed contribution relations initially to support requirements pre-traceability. "Pre-traceability is the ability to relate a requirement (called contribution) to the stakeholders that expressed and/or contributed to it (called contributors)" [Gotel95].

2.3 Different Approaches to the Generation of Traceability Links

This section describes the different levels of automation used to generate traceability links. It ranges from manual, to semi-automatic, and fully automatic generation. These

approaches are compared on the basis of amount of effort required to establish the traceability links, and the precision of these links.

2.3.1 Manual Generation of Traceability Links

In a manual process, a user manually identifies the artifacts to be traced. Traceability matrix is the classic way of establishing manual traceability relation. Another manual approach is by visualization and display tool components, in which the users identify and relate different elements from different traceable displayed documents. Tools that use such approaches of generation are Requirements Engineering Through Hypertext (*reth)* [Kozlenkov02], IBM's DOORS [TELEOLOGIC13], Integrated Chipware's RTM [RTM14] and Igatech's RDT [RDT].

Although the manual approaches provide users with an option to visualize and navigate through the generated traceability links, it is still a complicated effort, especially in the case of big and complex artifacts. The accuracy of the traceability links created manually depends on the user's understanding of the semantics of the links. Since the involved users can have different understanding, different discrepancies may occur while referring to the links.

2.3.2 Semi-Automatic Generation of Traceability Links

Semi- automatic approaches to generate traceability links have been introduced to overcome the problems related to the manual generation of traceability links.

According to Pohl there are development tools that may be used to generate semi-automatic traceability links. “This is a process-driven approach in which traceability links are generated because of creating, deleting, and modifying a product” [Pohl96].

Pohl also states that the development tools must ensure [Pohl96]:

a. “The recording of execution, input and output of each action related to the creation, deletion, and modification of a product in a trace repository”;
b. “Generation of dependency links between two dependent objects”, and
c. “Recording of the stakeholder performing the action and relationships between the action being executed and previous actions”.

Cleland-Huang suggested “an event-based approach to generate traceability links between requirements and performance models, and between non-functional requirements and design and code artifacts” [Cleland-Huang02, Cleland-Huang03]. As per Egyed, user defines traceability links during inception, elaboration and construction of the system [Egyed02].

These approaches may be recognized better than manual approaches; however, since user defines traceability link in some of the approaches they can be still be inaccurate, complex and time consuming.

2.3.3 Automatic Generation of Traceability Links

Some approaches have been proposed to remove user-defined relationships and automate the creation of traceability links. “Some of these approaches use information retrieval (IR) techniques” [Antoniol02, Hayes03, Marcus03], “others use traceability rules” [Ramesh92, Spanoudakis04, Krause03], “special integrators” [Sherba03], and “inference axioms” [Pinheiro00].

Antoniol, Hayes and Marcus proposed information retrieval techniques to create traceability links [Antoniol02, Hayes03, Marcus03]. Antoniol explained “traceability link between a requirement document and source code component” [Antoniol02]. This method assumes that “the vocabulary of the source code identifier overlaps with various items of the requirements documents as programmers usually choose names for their program items from the application-domain knowledge” [Antoniol02]. This method produces imprecise traceability links. Moreover, it only shows links between the artifacts that address the common features of a system [Antoniol02].

Hayes proposed vector space information retrieval (IR) techniques to automate the creation of traceability links [Hayes03]. This approach uses thesauruses or key-phrase list

to expand the classical vector IR model technique. “The study has shown that the use of a key-phrase list can improve the recall of the generated links (fewer missed links), but decreases their precision (i.e., it generates more irrelevant links) compared to classical vector IR techniques” [Hayes03].

Maletic proposed an approach “based on the use of *Latent Semantic Indexing* (LSI) to generate traceability links between system documentation (e.g., manual, requirements, design or test suites) and source code” [Maletic03]. This approach is independent of the programming language of the source code or language used in the system document. It considers synonymous terms by using “linear combinations of terms as dimensions of the representation space” [Marcus03]. In this approach, system documents and source code are pre-processed and a collection is created. “A traceability link between two documents is established when the semantic similarity measure of these documents is greater than a threshold” [Marcus03]. This approach has better recall and precision results compared to the two mentioned earlier.

Spanoudakis and Zisman proposed to use XML-base traceability rules to automatically create traceability links “between requirements statements, use cases, and analysis object” [Spanoudakis04, Zisman02]. “The documents to be traced are represented in XML and the generated links are represented as hyperlinks and expressed as an extension of Xlink” [DeRose01]. This approach has been evaluated in case studies for software-intensive TV systems and for a university course management system. The studies show

promising recall and precision levels ranging from 50% to 95%. These results back the automatic approach to automatically generate traceability links.

Ramesh proposed another approach to automatically generate traceability links between requirements and design artifacts in the Remap project. This approach is based on "trace rules and supports arbitrary chaining of rules in which the conclusion part of a rule can become part of the condition part of another rule" [Ramesh92]. Mohan has proposed an extension of Remap to establish traceability between customer requirements and design artifacts [Mohan02].

Based on relationship chaining Sherba explained an approach to generate new traceability links [Sherba03]. This approach uses special integrators that can detect and generate traceability links between software artifacts and other previously defined links. "The newly identified relations can be generated based on indirect and transitivity dependencies, complex dependencies containing more than one source or destination elements being related, or matching of pre-defined conditions between artifacts and/or links" [Sherba03]. User picks a specific chain of relation type when multiple chaining options are possible for some documents.

In Traceability of Object Oriented Requirement (TOOR), traceability links are defined and derived in terms of axioms. The tool uses these axioms to identify traceability links between requirements, design, and code specifications [Pinheiro96]. These axioms also enable users to define traceability links manually.

Most of these developed approaches (e.g. [Antoniol02, Hayes03, Marcus03]) have been implemented as prototypes and cannot completely automate the traceability links generation. They still have unacceptable precision levels and cannot support traceability for a big project. The approaches described by Gougen and Spanoudakis are "easy to use once a complete set of traceability relation generation rules and axioms have been identified" [Pinheiro96, Spanoudakis04]. But to establish those rules is not always easy.

Huergen Rilling, Rene Witte and Yonggang Zhang proposed an ontological approach to establish traceability links between software artifacts. Instead of using simple IR, they developed a Text Mining system for analyzing documents and a source code parser for the analysis of source code. The results from the source code and documents analysis are used to link the two software artifacts [Rilling07].

Cleland-Huang and colleagues have published several studies on IR-based trace recovery. They introduced probabilistic trace recovery using a PIN-based retrieval model, implemented in the tool Poirot. Their work focused on improving the accuracy of the tool by adding enhancements such as: "applying a thesaurus to deal with synonymy, extraction of key phrases, and using a project glossary to weigh the most important terms higher" [Cleland-Huang10].

Very recent work on IR-based trace recovery has gone beyond the traditional models for information retrieval [Borg13]. Several of these models are described below.

Hayes combined several IR models with a voting scheme mainly the probabilistic topic model Latent Dirachlet Allocation (LDA) [Hayes07]. Parvathy proposed use of Correlated Topic Model (CTM) [Parvathy08], and Gethers suggested use of Relational Topic Model (RTM) [Gethers11]. Abadi recommended the use of Probabilistic Latent Semantic Indexing (PLSI) that utilized two concepts based on information theory, Sufficient Dimensionality Reduction (SDR) and Jensen-Shannon Divergence (JS) [Abadi08].

In contrast, Capobianco proposed representing NL artifacts as B-splines and calculating similarities as distances between them on the Cartesian plane [Capobianco09]. Sultanov and Huffman Hayes implemented use of swarm technique to trace recovery. In this approach a non-centralized group of non-intelligent self-organized agents perform work which when combined, enables conclusions to be drawn [Sultanov10].

2.4 Different Approaches to the Representation, Recording, and Maintenance of Traceability Links

There are different architectural approaches to support the representation, recording and maintenance of traceability links in various tools and environments. These approaches can be differentiated into five categories based on "the level of integration between the artifacts and traceability relations, and the representation framework used to store the artifacts and relations" [Spanoudakis05]:

a. "The single centralized database approach"

b. "The software repository approach"

c. “The hypermedia approach”

d. “The mark-up approach”

e. “The event-based approach”

Different features and benefits of the above approaches to achieve traceability are discussed in the following sections.

2.4.1 Single Centralized Database Approach

The artifacts and the traceability link generated between them are both saved in a centralized database in this approach. Most of the requirement management applications that support traceability such as DOORS and RTM advocate this approach [TELEOLOGIC13, RTM14]. Usually, these applications save the traceability links between artifacts “in an underlying relational database” [Jarke01]. But, there are some tools such as TOOR that uses object-oriented database technology.

The underlying database provides user an advantage to efficiently query the traceability links. However, with this approach when an artifact is not created in the tool that manages the links, it is hard to generate and maintain traceability links between artifacts. Some applications such as IBM’s DOORS and RTM do provide artifact importing and exporting capabilities to alleviate this problem, however, importing and exporting is usually only possible for artifacts made by applications of the same company or by applications that work on a same framework. For example DOORS provides import and

export capabilities for artifacts that are created and managed by most of the popular CASE applications [TELEOLOGIC]. Nevertheless, import and export facilities are not effective in maintaining traceability links between evolving artifacts that are managed by different tools.

2.4.2 Software Repositories

Another approach is to use a centralized software repository and record traceability links along with the artifacts that they relate [Constantopoulos95, Pohl96]. The software repositories approach is different from the single database approach in a way that they are more flexible for defining schemas to store a variety of software artifacts and traceability link between them.

"The Software Information Base (SIB) is an experimental repository system that can support the definition of complex semantic structures for holding information about artifacts and traceability links at an infinite number of classification layers" [Constantopoulos95]. SIB is based on the data model of the conceptual modeling language TELOS. "It also allows the definition of arbitrary additional types of traceability relations, and provides an API through which it may be connected as an information server to external tools" [Constantopoulos95].

PRO-ART also uses the software repository approach [Pohl96]. PRO-ART is based on a process-centered approach. It integrates tools that are used to create, delete and modify an

artifact. These tools realize these actions and generate traceability links as a by-product when a user executes these actions. For instance, “when a developer creates a textual rationale for an object class, PRO-ART automatically creates a rationalization link between the class and the textual annotation” [Spanoudakis05]. In PRO-ART, the text editor may create the text providing the rationale, which is different from the tool that created the linked class. The huge advantage of process-centric approach is that it enables user to achieve traceability during the software development processes [Dogmes98]. However, in the beginning the software repository approach needs rigorous tool integration. This may not be possible in distributed software development settings.

2.4.3 The Hypermedia Approach

The hypermedia approach is based on open hypermedia architecture. It is proposed as a solution to maintain traceability links between evolving artifacts without integrating the relevant tools around a software repository. [Whitehead97].

Using this approach Sherba has developed a traceability management tool, called TraceM [Sherba03]. “TraceM is a research prototype system that supports the recording, maintenance, and traversal of links between software artifacts that are constructed by heterogeneous tools” [Sherba03]. TraceM stores traceability links independent of the artifacts that they relate. In TraceM a traceability link can use metadata to define an n-ary relation between artifacts, or their parts. These metadata specify:

a. “The types of the artifacts associated by a link

b. The external tools that create these artifacts
c. Transformers that can be used to transform artifacts into the common representation framework of TraceM
d. Integrators that can be used to automatically discover and create the traceability links" [Sherba03].

Metadata can also specify the different stakeholders interested in a project. To completely utilize TraceM services, external tools that are used to create artifacts have to be integrated with TraceM. This integration can be accomplished using standard techniques available in open hypermedia environments. TraceM can also be used to only document and view links when non-integrated external tools are used to create artifacts [Sherba03].

2.4.4 The Mark-Up Approach

The use of markup language is proposed to achieve traceability in widely distributed and heterogeneous software engineering settings. In this approach the traceability links are saved independent of the artifacts that they relate.

"Gotel and Finkelstein have developed a toolkit that can be used to create and maintain contribution relations. This toolkit uses a combination of HTML and descriptive mark up representations to store different types of contribution relations between artifact's as hyperlinks" [Gotel95]. Maletic has also developed a tool that saves traceability links as different types of "hyperlinks" [Maletic03].

“STAR-Track is a web-based requirements tracing tool that uses tagging mechanisms to represent traceability links” [Song98]. The tags in STAR-Track show artifacts or links between these artifacts. Every tag includes a title and an artifact identifier.

In the rule-based tracer Spanoudakis, et al. described that both artifacts and traceability links that relate them are represented in XML [Spanoudakis04]. However, traceability artifacts are stored separately from their traceability links and XLink elements are used to mark the parts of the artifacts these links relate [DeRose01]. This system can also convert textual artifacts from their initial format into XML. One main advantage of the rule-based tracer is that it does not need any kind of tool integration. However, it provides limited support to maintain traceability links of evolving artifacts [Spanoudakis05]. This is because the modified artifacts need to be re-transformed into XML to ensure traceability link remains valid after the changes.

2.4.5 The Event-Based Approach

In all above approaches (excluding the centralized software repository approach), it is difficult to maintain the traceability links between evolving artifacts. To solve this problem, Cleland-Huang “developed an event-based traceability (EBT) server to document and maintain traceability links between requirements documents and other software artifacts” [Cleland-Huang02]. EBT is based on a registry system where the requirement documents can register their dependencies to other artifacts, and the system monitors the artifacts after the registration of dependencies. It informs all dependent

requirements when an artifact is changed. After the dependency relations are recognized this system makes it easier to maintain them. However, the system does not recognize those relations.

Centralized database and the software repository approach have better performance and data management facilities than even-based and mark-up approaches. But, the latter are more interoperable.

2.5 Different Ways of deploying Traceability Links

Traceability links may be used to support different development and maintenance activities in the development life cycle of a software system including:

a. "Change impact analysis and management" [Cleland-Huang02]
b. "System verification, validation, testing and standards compliance analysis" [Jarke01]
c. "The reuse of software artifacts" [Paech02]
d. "Software artifacts understanding" [Antoniol02, Marcus03, Ramesh92, Jarke01].

The following sections describe how traceability may support the above activities and explain different traceability links that can be used for that.

2.5.1 Traceability for Change Impact Analysis and Change Management

One of the main reasons to achieve traceability between different artifacts is the ability to use the traceability links for:

a. Change Impact Analysis - Establish the impact of potential changes to the system.
b. Change Management - Decide whether these changes should be introduced.

The easiest way to analyze the impact a change in an artifact will cause (e.g. a use case) is by identifying of all the other artifacts that will be affected by the change (e.g. source code and test case). Basic querying abilities must exist in a system for simple change impact analysis [Spanoudakis05]. Many of the current traceability tools and environments provide ability to query and retrieve traceability links between particular types of artifacts. However, some systems can require more complex way of change impact analysis. Some of these ways are:

a. The distributions of impacted artifacts into various groups due to the specific effect caused by the change;
b. The identification of any side-effect because of the change;
c. The estimation of the cost due to the change.

These capabilities require arranging different traceability links into trace-paths. These trace-paths can show how changes in an artifact may impact artifacts that are not directly linked. Structures of traceability links may be established by “evaluating regular expressions” [Pinheiro96], “deductive rules” [Jarke01], or “traceability rules”

[Spanoudakis04]. Few research models such as TOOR [Pinheiro96], PRO-ART [Pohl96], and the rule-based tracer [Spanoudakis04] support similar structure capabilities. However, many of the industrial traceability tools generate appropriate scripts to estimates these compositions. Moreover, it does not provide significant cost estimation of executing and integrating requested changes. This is because the "few cost estimation models that have been developed for this purpose cannot provide very precise cost predictions" [Lavazza00].

Certain systems may execute a simulation model to assess impact of a potential change. The EBT system is capable of analyzing "the effect of requirement changes onto the performance of software systems by using dependency relations between requirements and performance simulation models" [Cleland-Huang02].

Certainly, the semantics, granularity and accuracy of traceability links determine the accuracy of both simple and complex forms of impact analysis. Even though the links that are generated by identifying references to common entities in different software artifact may support the impact of a change on an artifact. It still cannot establish or measure this impact with certainty. Examples of such artifact links are "the overlap relations between requirements and object-oriented analysis models" identified by Spanoudakis, et al. [Spanoudakis04], or "between source code artifacts and manual pages or requirements identified by information retrieval techniques" by Antoniol and Marcus [Antoniol02, Marcus03]. However, traceability link with a rich semantic content (e.g. the dependency relations [Jarke01]) can provide more accurate impact analysis results.

Studies have shown the need of traceability link with rich semantics to measure accurate impact analysis [Bianchi00, Lindval96]. Empirical research has also shown that granularity of artifacts that are linked defines accuracy of impact predictions. Bianchi reported that "fine-grain links that relate specific artifacts/parts within broad models and documentation result in more accurate results" [Bianchi00]. Different case studies in software industry have also shown that software engineers do not trust the automatically generated traceability links due to doubts about the accuracy of these links [Lindval96].

2.5.2 Traceability for Software Validation, Verification, Testing and Standards Compliance

Traceability links can provide various methods to analyze and ensure that "a system implements the requirements provided by the various stakeholders involved (validation), verify that it satisfies certain properties and its specification (verification), test its sole elements and the system as a whole, and evaluates that it meets existing standards" [Spanoudakis05].

"Pre-traceability contribution relations, for instance, may be used to identify stakeholders and involve them in requirement validation activities" [Gotel95]. Similarly, traceability links may be used to ensure that test cases exist to verify every requirement. "The results of this preliminary verification analysis typically provide input to software inspection and auditing procedures" [Paech02].

Traceability links can also be used to assess the consistency of different models of a system [Eastbrook98]. For example, Spanoudakis presents an approach to convey *overlap* relations that holds common feature. He suggests rewriting formal requirement specifications to check their consistency using theorem proving [Spanoudakis99]. Fiutem and Antoniol used "string-matching algorithms to extract design models from the source code of a system" [Fiutem98]. After extracting they discover overlap relationship between these models and the initial design model that was developed before implementation. Later these relationships are used "to verify the consistency of the implementation with the original design" [Fiutem98].

2.5.3 Traceability for Software Reuse

Traceability links have been widely used to identify reusable artifacts in the software development life-cycle [Antoniol02, Constantopoulos95, Paech02]. Using these links the artifacts at different levels of abstraction (e.g. source code, design or requirement artifacts) can be identified and reused through different scenarios.

"Constantopoulos used *dependency* traceability relations, called correspondence relations to associate requirements specifications with design models, and design models with source code in the *Software Information Base*" [Constantopoulos95]. Following these relations in an application, software engineers can find concrete reusable artifacts at low levels of abstraction (design and source code artifacts). Dick and NASA have suggested similar approaches [Dick02, NASA]. In NASA, "reusable design elements are identified

through satisfiability, refinement or overlap relations that connect them with events, pre-conditions, and post-conditions in use case models" [NASA]. Antoniol has also recommended use of traceability links between source code artifacts (e.g. functions in code libraries) and manual pages to enable software engineers to reuse these artifacts in specific contexts [Antoniol02].

2.5.4 Traceability for Artifact Understanding

Tractability links can also be used to understand the reason behind the creation of the involved artifacts or the relationship between them. Understanding these reasons can be very useful in scenarios where the people who need to access and maintain the artifacts are not the ones who created them, a common phenomenon in software maintenance. For instance "objective of enabling code comprehension has driven the development of approaches that can trace components of programs generated by deductive synthesis (e.g. variable names, function calls) onto the specifications from which they were derived" [Cleland-Huang03].

Similarly, rationalization relations can be used to explain about the form of requirement and design artifacts [Pohl96, Ramesh92, Jarke01]. "PRO-ART and REMAP are examples of the traceability environments that support the recording of such relations" [Ramesh92]. This model can be used to represent the issues that were recognized during the construction of the artifacts.

In summary, this chapter provided an overview of the research and practical work completed in the field of software system traceability. This chapter showed importance of establishing traceability in the software development life cycle. Even though the current methods have made important improvements to different aspects of traceability it is still a difficult task. Looking at the current state of research and available tools traceability is not extensively used in the industry [Spanoudakis05]. After looking at the roadmap of research and practical work in software system traceability the next section provides an overview of the need for traceability in the software project and tools used in my research project.

2.6 Need for Traceability

Research has shown that insufficient traceability is one of the leading causes of software project failures and budget overruns [Domges08]. As a result, considerable research has been undertaken on this topic recently and many organizations are endeavoring to improve their traceability practices [Kannenberg09]. The results of these efforts can be seen in Standish Group's 2013 CHAOS manifesto [CHAOS13]. "The 2012 CHAOS results show an increase in project success rate of 39% compared to a success rate of 37% in 2010 (delivered on time, on budget, with required features and functions); and decrease in number of failures of 18% in comparison to 21% in 2010 (cancelled prior to completion or delivered and never used)" (refer to Table 6) [CHAOS13].

	Successful	Failed	Challenged
2004	29%	18%	53%
2006	35%	19%	46%
2008	32%	24%	44%
2010	37 %	21%	42%
2012	39 %	18%	43%

Table 6. Project Resolution Results from CHAOS Research for years 2004 to 2012 [CHAOS13]

Kannenberg, et al. suggests that the software engineering industry accepts the importance of traceability but principles of traceability are still not well understood in many organizations [Kannenberg09].

2.7 Traceability in the Research Project

Given the past history of traceability as addressed by many researchers over recent years, coupled with genuine acknowledged need for traceability in the production of modern application data systems, this research will address traceability for small, real-world software development projects within a modern application development framework using a commonly-accepted development methodology.

Specifically, traceability will be applied to the design and development of the Value Adjustment Board Project (VAB) of City of Jacksonville as needed by the Duval County property appraiser's office. Using the Scrum development approach within Rational Team Concert (RTC), traceability will be demonstrated throughout life-cycle activities.

Chapter 3

METHODOLOGY

3.1 Application Lifecycle Management

"Application lifecycle management (ALM) is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management" [Johnson11]. By the end of the 1970's, researchers like Yourdon, Orr, Dijkstra, and DeMarco began discussing and creating functionally strong software applications, strengthening the field in sound new ways [Wood 10].

Yourdan utilized Date Flow Diagrams (DFDs), providing a process-oriented method that proved extremely useful for data capture, and was thus used by many in the field for years to come. From here, Software Engineering (CASE) tools emerged from varied methodologies that began to support traceability [Wood10].

3.2 Collaborative Lifecycle Management

"Collaborative Lifecycle Management (CLM) is an integrated Application Lifecycle Management solution by IBM that integrates different products based on Jazz™

framework" [Gothe08]. It connects the artifacts of analysts with artifacts of developers and testers. "Jazz™ is built on architectural principles that represent a key departure from approaches taken in the past. Together, these approaches allow teams to surf the collaborative Web to seamlessly access teams, processes and artifacts" [Jazz12]. "CLM links among the products support traceability, web-like navigation, review, commenting, and status tracking across various project repositories" [Gothe08]. According to Babcock, CLM consists of three main applications: "IBM Rational Requirements Composer, IBM Rational Team Concert, and IBM Rational Quality Manager" [Babcock12].

Rational Requirement Composer (RRC) is an application used to define and manage requirements for any size of project. "Project teams can manage their requirements, write good use cases, improve traceability, strengthen collaboration, reduce project risk, and increase quality" [Gothe08].

Rational Team Concert (RTC) is a Rational product on the Jazz™ team collaboration platform. It was initially released on 2008. RTC provides various abilities to integrate work of team members with different roles:

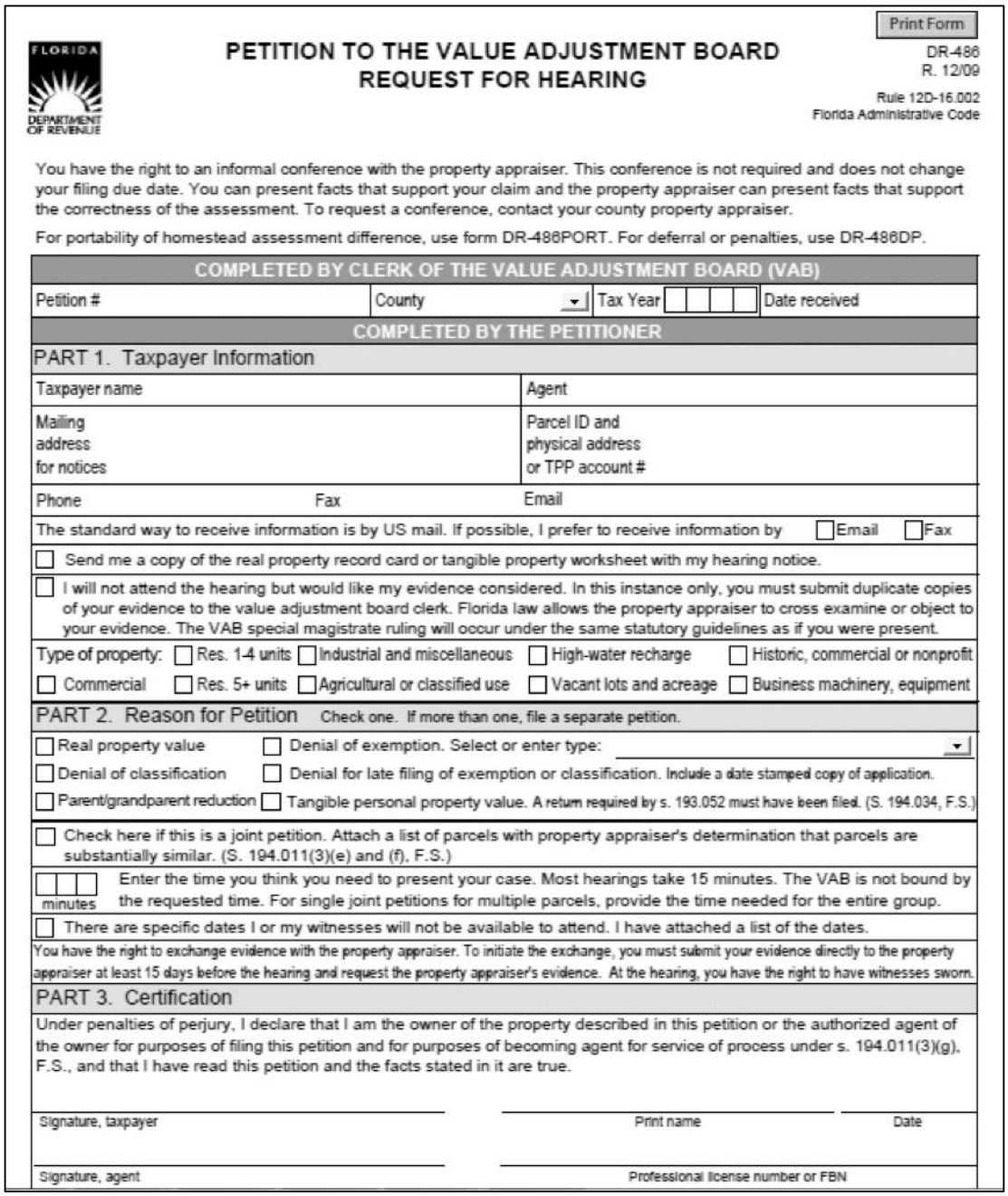
a. Seamlessly integrates different tasks through the software life cycle;
b. Provides better team collaboration to develop more effective applications;
c. Generates and maintain traceability links to automate the bookkeeping [Gothe08].

Rational Quality Manager (RQM) is another application on Jazz™ team collaboration platform. RQM is a complete test management system that provides testers to plan, construct and execute tests in RQM. [Gothe08].

3.3 Project

Using the Scrum development approach within IBM tools on Jazz framework - RRC, RTC and RQM - this research presents a case study in traceability as applied to the design and development of the Value Adjustment Board Project (VAB) of City of Jacksonville as needed by the Duval County Property Appraiser's Office.

“The Value Adjustment Board reviews appeals from decisions made by the Duval County Property Appraiser. VAB jurisdiction includes appeals of property value assessments, exemption denials, agricultural (greenbelt) classification denials, and portability appeals, among others” [VAB14]. The current appeal process is done manually by submitting the DR-486 form to the Clerk of the VAB at the Property Appraiser Office in person or in mail along with the filing fee.



Print Form

DR-486
R. 12/09

Rule 12D-16.002
Florida Administrative Code

FLORIDA DEPARTMENT OF REVENUE

PETITION TO THE VALUE ADJUSTMENT BOARD
REQUEST FOR HEARING

You have the right to an informal conference with the property appraiser. This conference is not required and does not change your filing due date. You can present facts that support your claim and the property appraiser can present facts that support the correctness of the assessment. To request a conference, contact your county property appraiser.

For portability of homestead assessment difference, use form DR-486PORT. For deferral or penalties, use DR-486DP.

COMPLETED BY CLERK OF THE VALUE ADJUSTMENT BOARD (VAB)

Petition #	County	Tax Year	Date received

COMPLETED BY THE PETITIONER

PART 1. Taxpayer Information

Taxpayer name	Agent
Mailing address for notices	Parcel ID and physical address or TPP account #
Phone Fax	Email

The standard way to receive information is by US mail. If possible, I prefer to receive information by ☐ Email ☐ Fax

☐ Send me a copy of the real property record card or tangible property worksheet with my hearing notice.

☐ I will not attend the hearing but would like my evidence considered. In this instance only, you must submit duplicate copies of your evidence to the value adjustment board clerk. Florida law allows the property appraiser to cross examine or object to your evidence. The VAB special magistrate ruling will occur under the same statutory guidelines as if you were present.

Type of property: ☐ Res. 1-4 units ☐ Industrial and miscellaneous ☐ High-water recharge ☐ Historic, commercial or nonprofit
☐ Commercial ☐ Res. 5+ units ☐ Agricultural or classified use ☐ Vacant lots and acreage ☐ Business machinery, equipment

PART 2. Reason for Petition Check one. If more than one, file a separate petition.

☐ Real property value ☐ Denial of exemption. Select or enter type:
☐ Denial of classification ☐ Denial for late filing of exemption or classification. Include a date stamped copy of application.
☐ Parent/grandparent reduction ☐ Tangible personal property value. A return required by s. 193.052 must have been filed. (S. 194.034, F.S.)

☐ Check here if this is a joint petition. Attach a list of parcels with property appraiser's determination that parcels are substantially similar. (S. 194.011(3)(e) and (f), F.S.)

☐☐☐ minutes Enter the time you think you need to present your case. Most hearings take 15 minutes. The VAB is not bound by the requested time. For single joint petitions for multiple parcels, provide the time needed for the entire group.

☐ There are specific dates I or my witnesses will not be available to attend. I have attached a list of the dates.

You have the right to exchange evidence with the property appraiser. To initiate the exchange, you must submit your evidence directly to the property appraiser at least 15 days before the hearing and request the property appraiser's evidence. At the hearing, you have the right to have witnesses sworn.

PART 3. Certification

Under penalties of perjury, I declare that I am the owner of the property described in this petition or the authorized agent of the owner for purposes of filing this petition and for purposes of becoming agent for service of process under s. 194.011(3)(g), F.S., and that I have read this petition and the facts stated in it are true.

Signature, taxpayer	Print name	Date
Signature, agent	Professional license number or FBN	

Figure 9. DR 486 Form

This research will use the CLM framework to take the small team project from needs to features, features to use cases, use cases to design documents and design documents to

test cases. The research will demonstrate the support environment for tracking and ensuring consistency in the above flow.

3.4 Project Constraints

The project was part of the graduate Software Engineering I and II courses at the University of North Florida, so it was completed under the following constraints:

- The project has five team members – one business analyst, two developers and two testers to implement within a school year.
- The team uses Scrum and Unified Processing methodologies to manage the project.
- The team work is confined to a regular, repeatable work cycle, known as a sprint
- The team submits a deliverable (working product along with the documentation requested in the class) after each sprint
- The goal of the project is to create two web applications to expedite the VAB appeal process.
- Application uses Microsoft SQL Server for database, Visual Studio 2012 for the integrated development environment (IDE) and C# for the programming language.

Chapter 4

RESEARCH RESULTS

The artifact linkages shown in Figure 10 were used on the VAB project for COJ to achieve traceability. Rational Team Concert (RTC) implements requirements from Rational Requirements Composer (RRC) and creates a linkage among them. Similarly, Rational Quality Manager (RQM) validates the requirements in Rational Requirements Composer and forms a linkage between the two project areas. In the end, Rational Quality Manager tests work items in Rational Team Concert and creates a linkage between these two project areas.

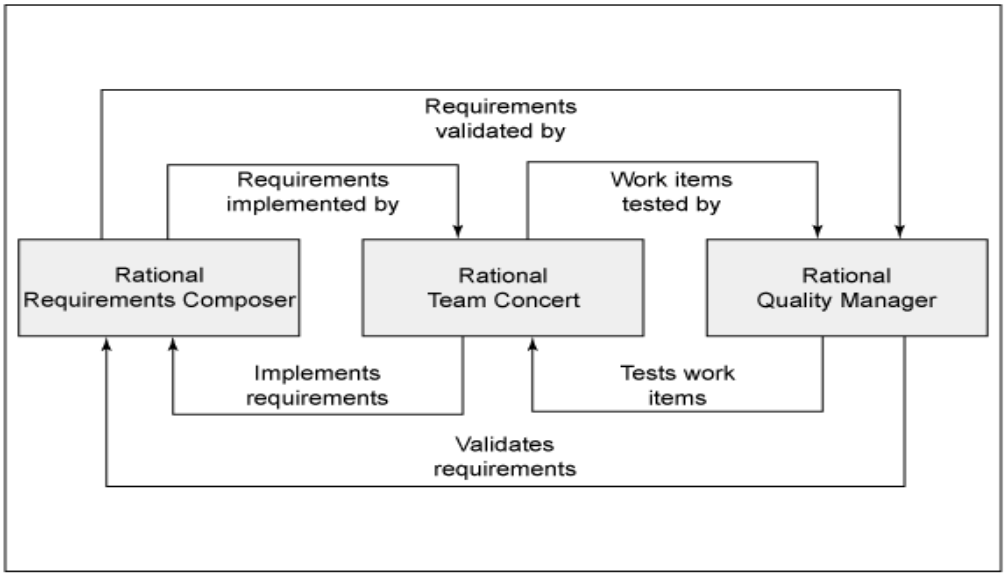


Figure 10. Artifact Linkage across Rational Requirements Composer, Rational Team Concert, and Rational Quality Manager [DEV10]

4.1 Linking Project Areas in Rational Requirements Composer and Rational Team Concert

The first step to link project areas as shown in Figure 10 is to link Rational Requirement Composer (RRC) and Rational Team Concert (RTC). For this, "COJ Value Adjustment Board Project (Requirements)" project area in RRC and "COJ Value Adjustment Board Project (Change Management)" in RTC were linked by taking the following steps.

1. Log on to the server administration interface of Rational Requirements Composer server as an Admin group member (user with administrative privileges).
2. Navigate to the project overview page of the "COJ Value Adjustment Board Project (Requirements)" (refer to Figure 11) and click the Add button in the Associations section (refer to Figure 12).

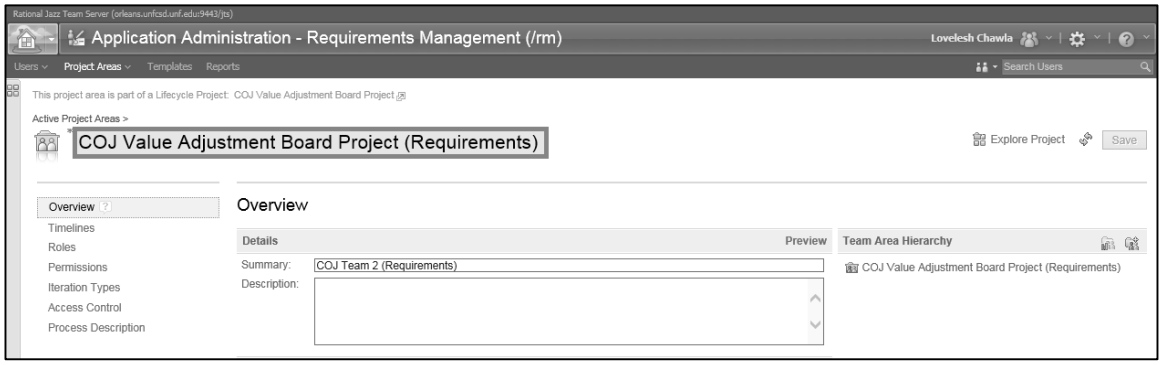


Figure 11. COJ Value Adjustment Board Project (Requirements) Overview Page

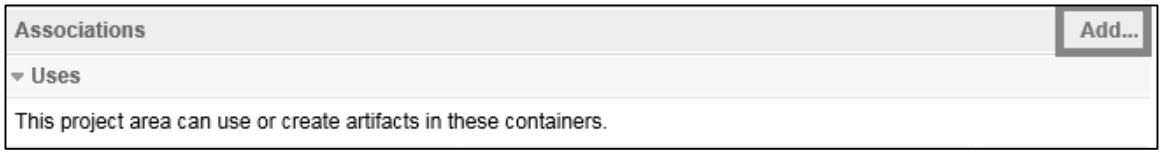


Figure 12. Add Button in the Associations Section

3. Under the Applications select "/ccm", under Associations select "Uses – Requirements Change Requests", and under Artifact Containers select "COJ Value Adjustment Board Project (Change Management)" (refer to Figure 13).

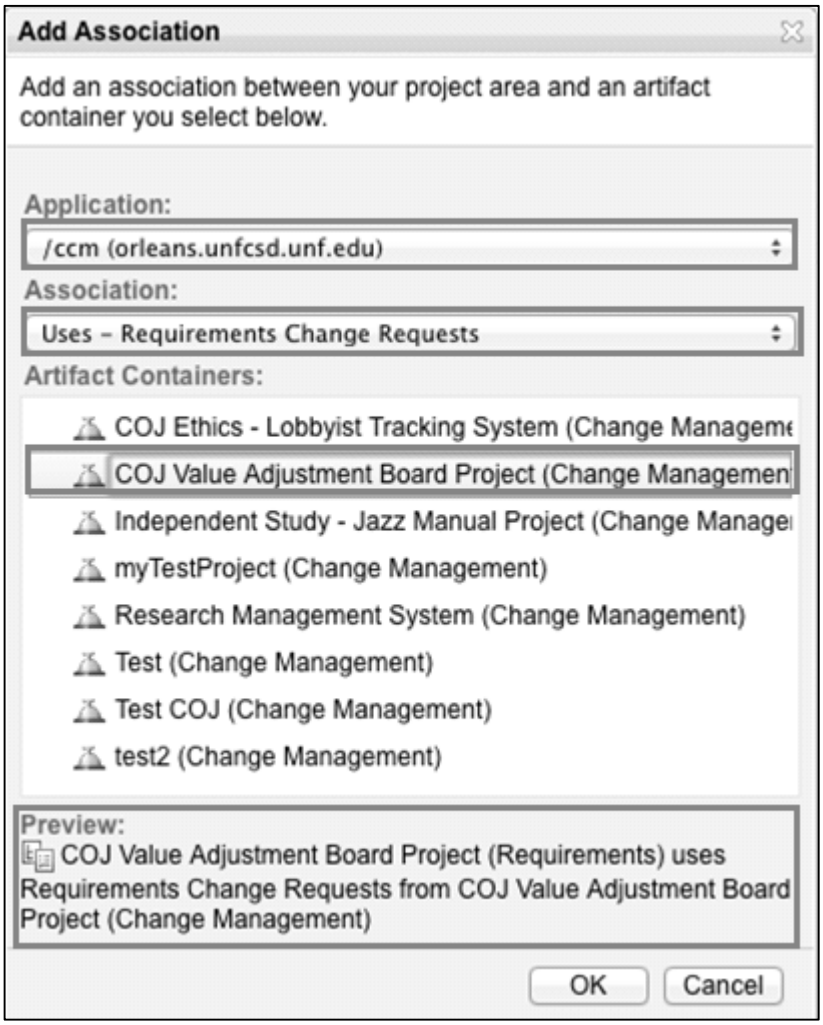


Figure 13. Linking COJ Value Adjustment Board Project (Requirements) with COJ Value Adjustment Board Project (Change Management)

The linkage between" COJ VAB Project (Requirements)" and "COJ VAB Project (Change Management)" can be verified by the link created in the "uses" section of the

project area overview page of "COJ Value Adjustment Board Project" in Rational Requirement Composer (refer to Figure 14).

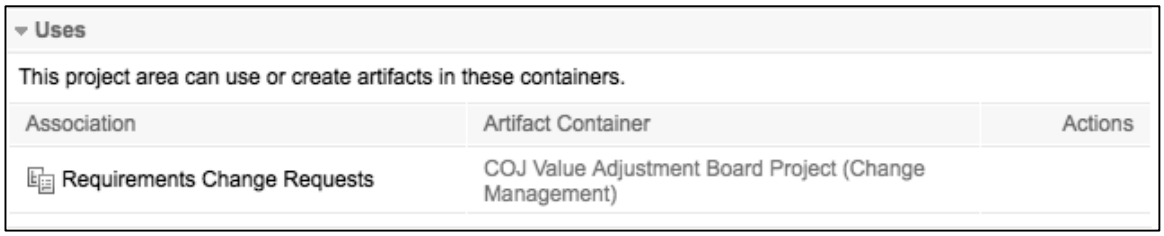


Figure 14. Uses Section of the Project Area Overview Page of "COJ Value Adjustment Board Project"

After establishing project linkage, the next step is to link the artifacts.

A requirement in Rational Requirements Composer and a work item in Rational Team Concert are used to establish the link between Rational Requirements Composer and Rational Team Concert.

4.1.1 Linking Rational Team Concert Work Items to the Requirements in Rational Requirement Composer

Requirements are defined in Rational Requirement Composer under “COJ VAB Project (Requirement)” for each sprint; that is, the complete project divided into time-boxed efforts each of a specific duration.

The following requirements from the COJ VAB are now shown in Figure 15 are part of the Deliverable 1; that is, documentation (use cases in this case) required at the end of

sprint 1 (refer to Figure 15)

1. Submit Petition
2. Withdraw Petition
3. View Petition
4. Update Petition

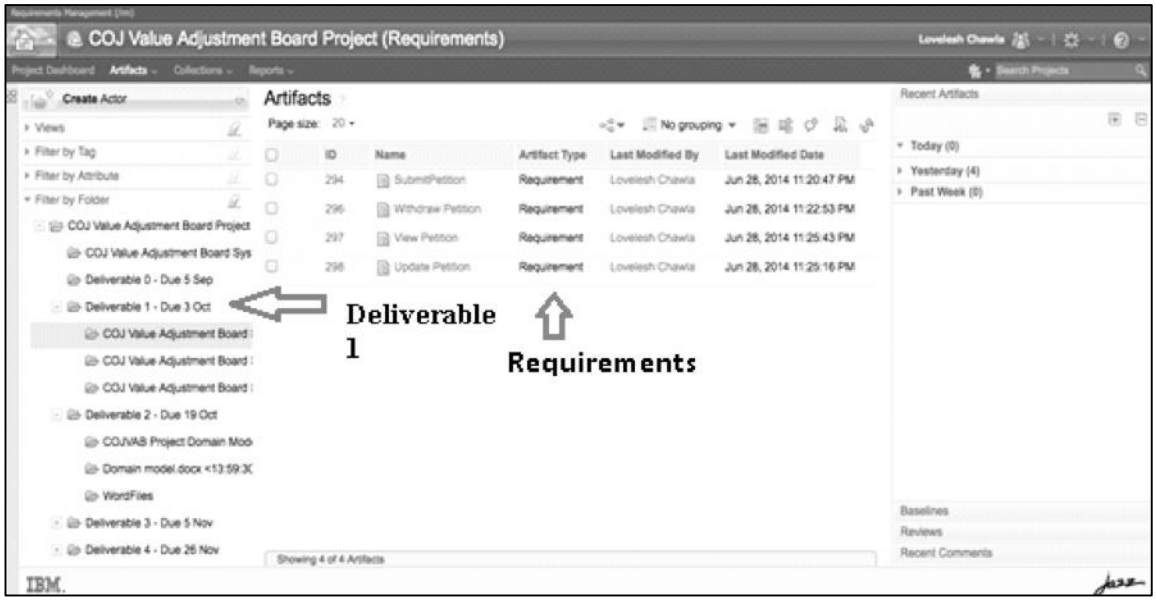


Figure 15. Requirements for Deliverable 1

The following steps create a Work Item (fundamental mechanism to track and coordinate development tasks and workflows in Rational Team Concert) that was created and shown in Figure 15. These steps are used to link each work item of the "COJ VAB Project (Change Management)" with a requirement in "COJ VAB Project (Requirement)":

1. Using the Rational Team Concert web UI, navigate to the work Items section of the COJ VAB Project (Change Management) (refer to Figure 16).

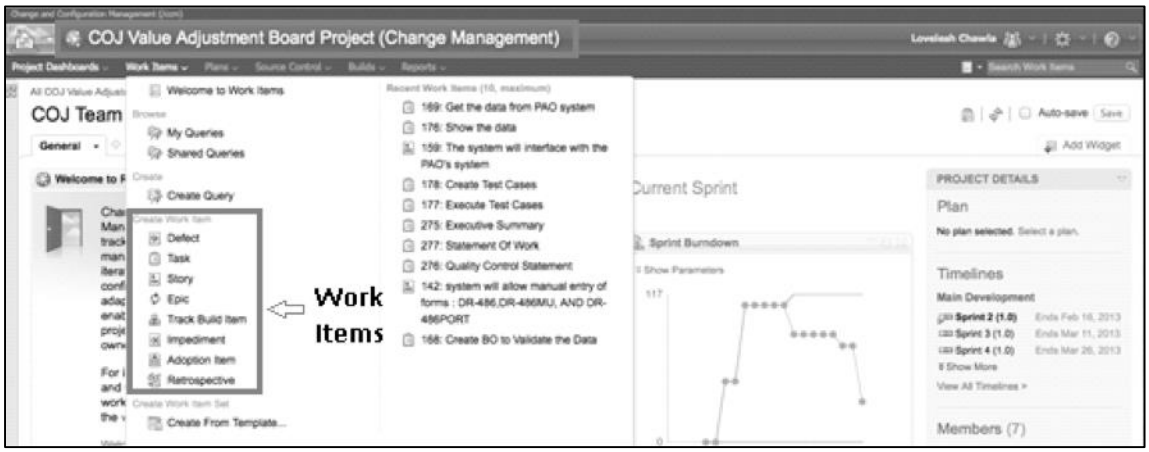


Figure 16. Work Items Section of COJ VAB Project (Change Management)

2. Choose a Work Item from the available work items such as Task, Defect, Story etc.
3. Fill the Task information in the overview section (refer to Figure 17).

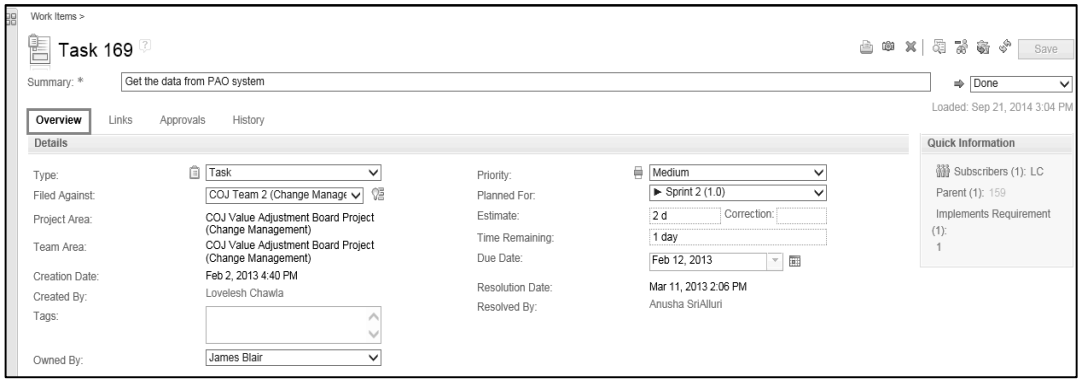


Figure 17. Overview Section of Task

4. Click on the links tab and choose “Implements Requirement” from the different links dropdown (refer to Figure 18).

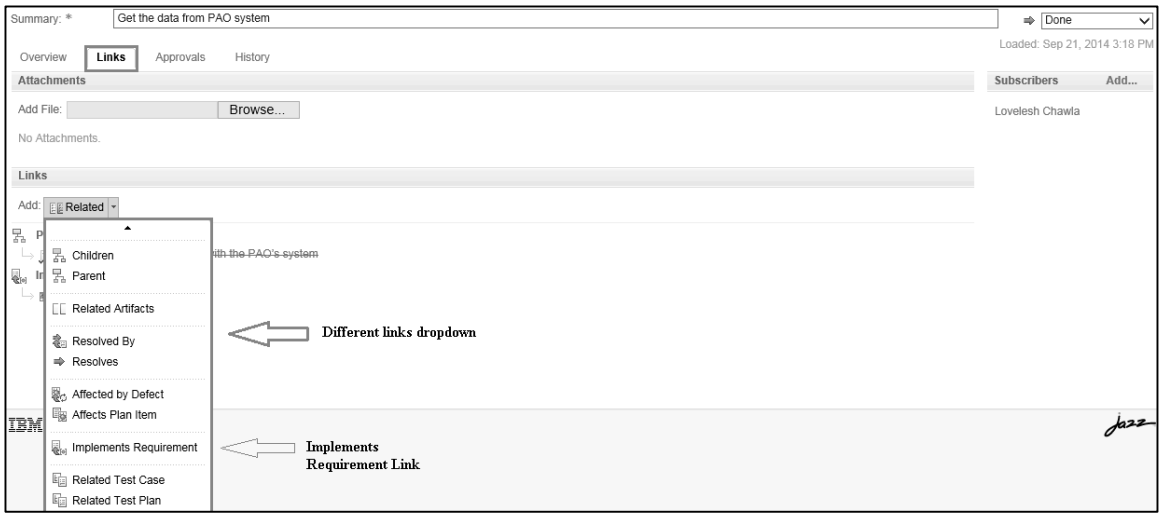


Figure 18. Links Tab

5. Choose COJ Value Adjustment Board Project (Requirements) under the location of artifact and pick link to existing requirement (refer to Figure 19).

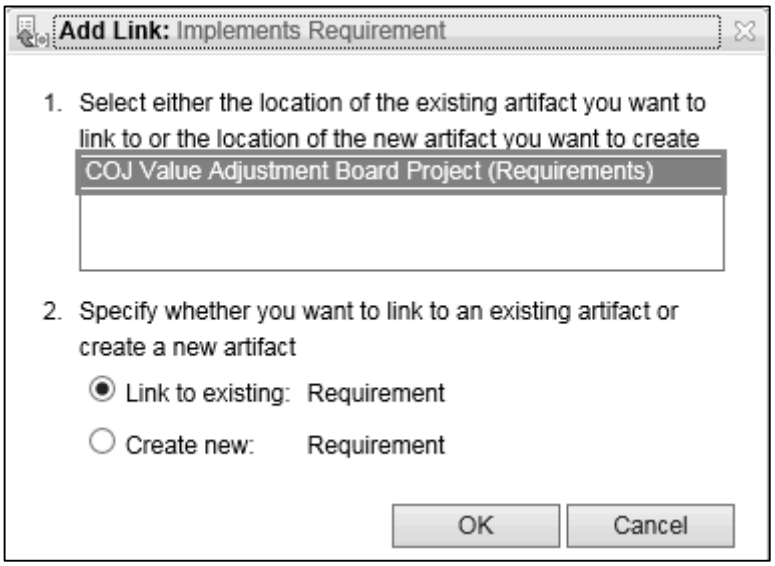


Figure 19. Link Related Track Requirement for new Work Item

6. Next step is to choose a requirement from the above requirements (refer to Figure 20).

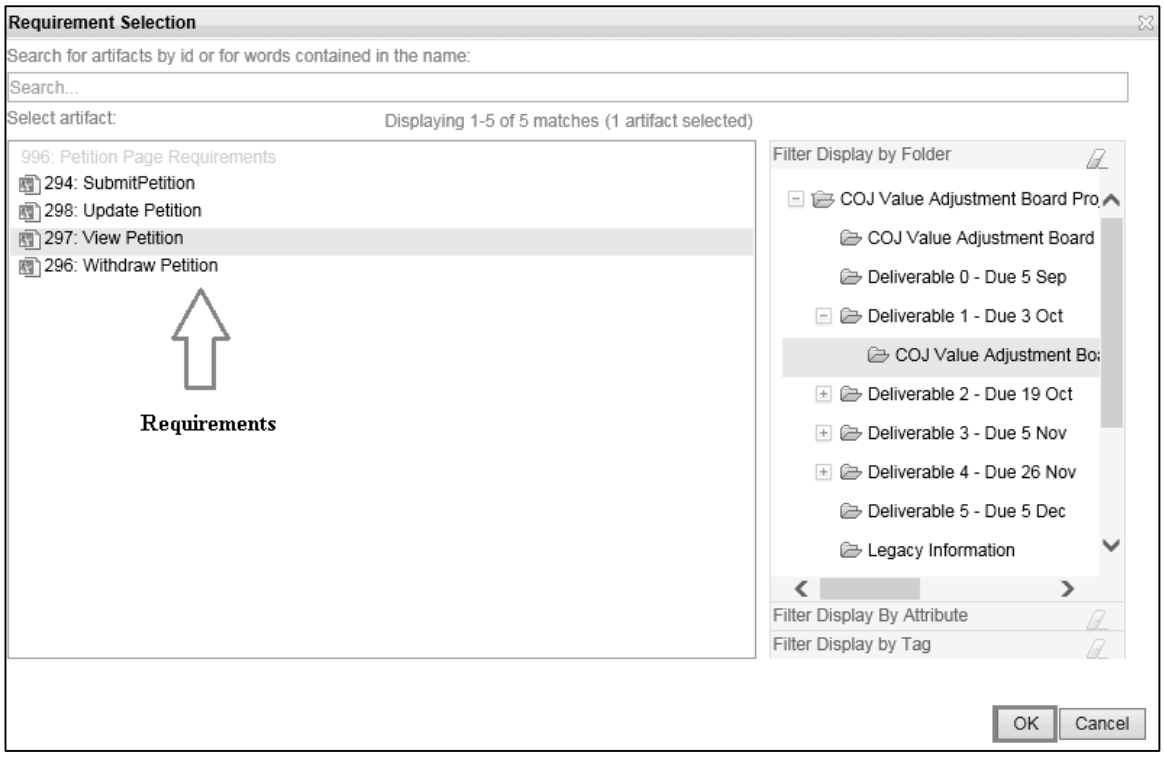


Figure 20. Requirement Selection Popup for Work Item in Rational Team Concert

The newly added link between the work item and requirement can be verified after selecting OK in the links section (refer to Figure 21).

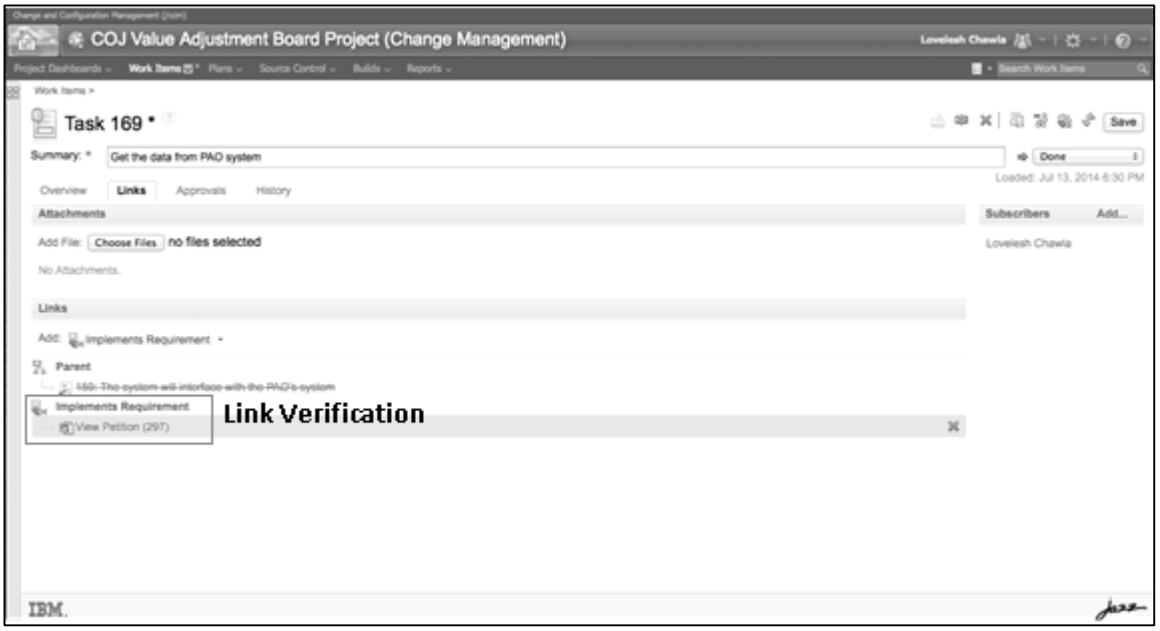


Figure 21. Requirement linked to Work Item

These steps were repeatedly followed to link all the requirements from COJ VAB Project (Requirements) in RRC to work items from COJ VAB Project (Change Management) in RTC and thus creating traceability between requirements of the project and tasks of the project.

4.2 Linking between Project Areas in Rational Requirements Composer and Rational Quality Manager

In the previous section the relationship between RRC and RTC was established. This section shows how the link between the COJ VAB Project (Requirements) in RRC and COJ Value Adjustment Board Project (Quality Management) in Rational Quality Manager was created.

1. Log on to the server administration interface of Rational Requirements Composer

server as an Admin group member (user with administrative privileges).

2. Navigate to the project overview page of the COJ Value Adjustment Board Project (Requirements) and click the Add button in the Associations section (refer to Figure 22).

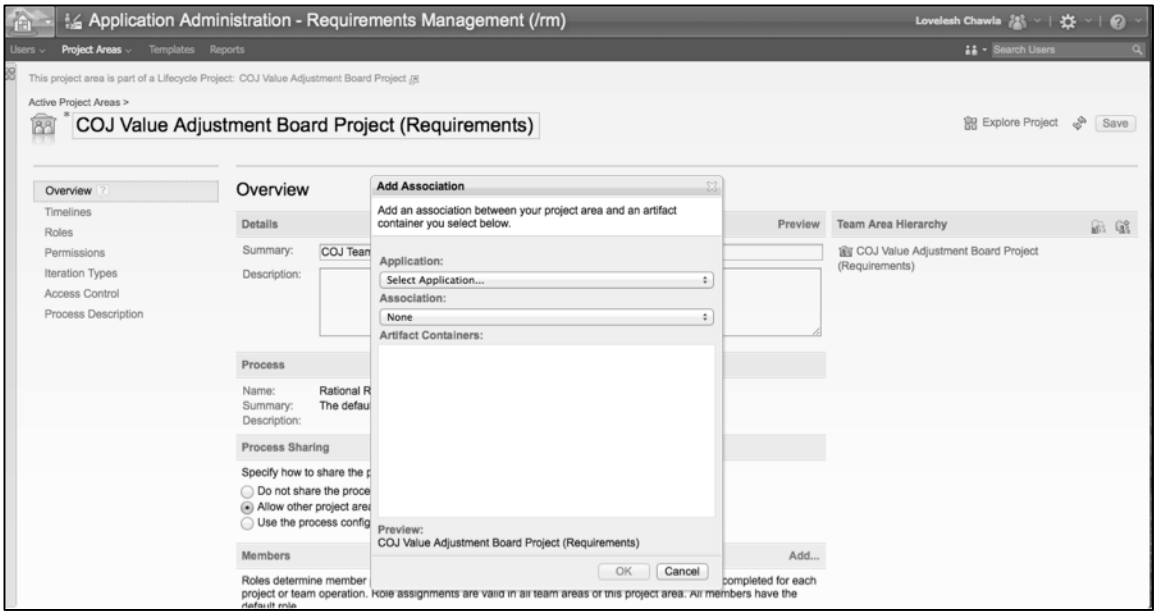


Figure 22. COJ Value Adjustment Board Project (Requirements) Overview Page

3. Under Applications select “/qm”, under Associations select “Provides – Requirements” and under Artifact Containers select COJ Value Adjustment Board Project (Quality Management) (refer to Figure 23).

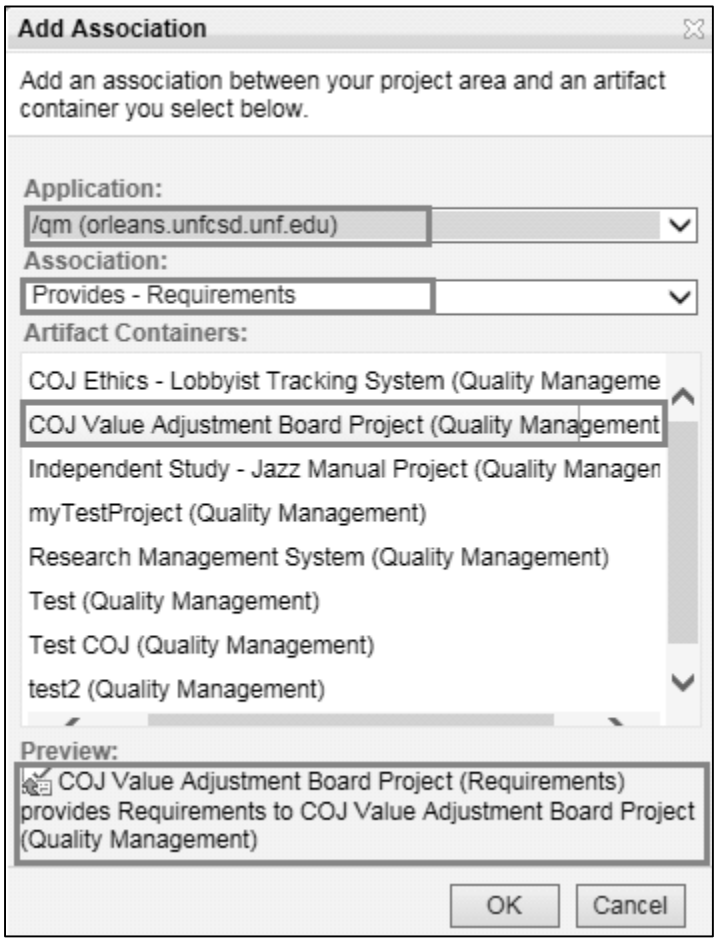


Figure 23. Linking COJ Value Adjustment Board Project (Requirements) with COJ Value Adjustment Board Project (Quality Management)

The linkage between "COJ VAB Project (Requirements)" and "COJ VAB Project (Quality Management)" can be verified by the link created in the "provides" section of the project area overview page of COJ Value Adjustment Board Project in Rational Requirement Composer (refer to Figure 24).

Figure 24. Provides Section of the Project Area Overview Page of "COJ Value Adjustment Board Project"

4.2.1 Linking Test Cases in Rational Quality Manager to the Requirements in Rational Requirements Composer

A requirement in COJ Value Adjustment Board Project (Requirements) is linked with a test case in COJ Value Adjustment Board Project (Quality Management)

The following steps were performed to link a requirement with a test case:

1. Using the Rational Quality Manager web UI a test case was created in COJ VAB Project (Quality Management) (refer to Figure 25).

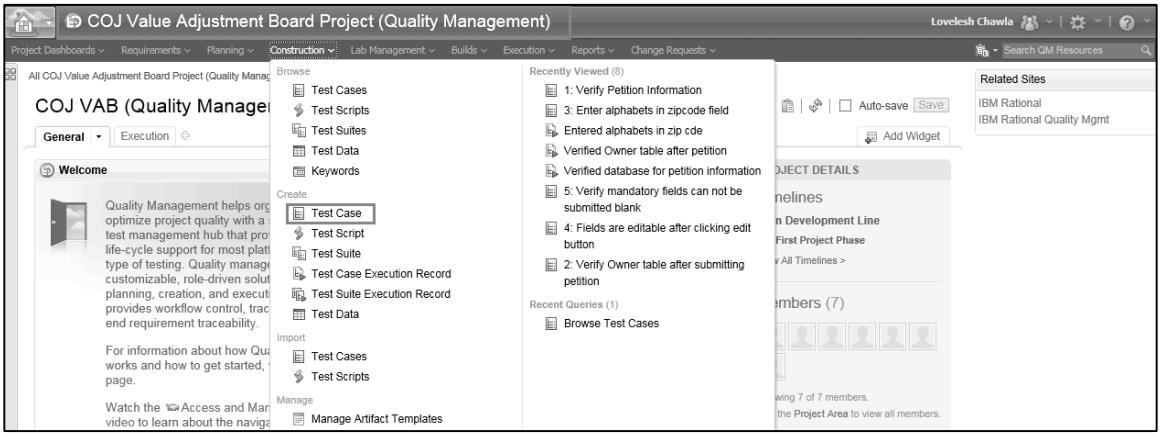


Figure 25. Rational Quality Manager Web UI for COJ VAB Project (Change Management)

2. All the details for the test case were filled in the summary section (refer to Figure 26).

Figure 26. Summary Section of Test Case

3. A link to a requirement was added by going to the "Requirement Links" section of the test case and clicking the "Add new link" button (refer to Figure 27).

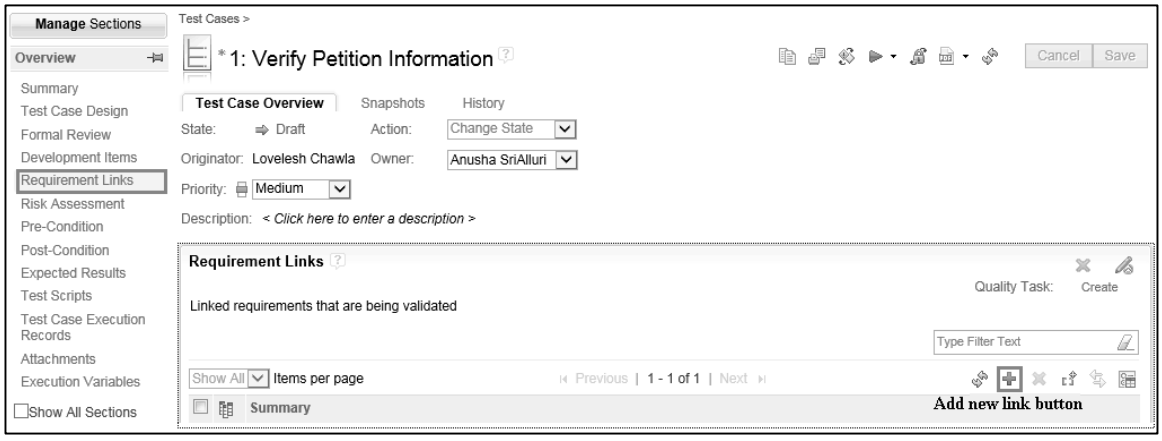


Figure 27. Requirement Links Section

4. Chose the requirement “View Petition” to link to the test case and click “Ok” button (refer to Figure 28).

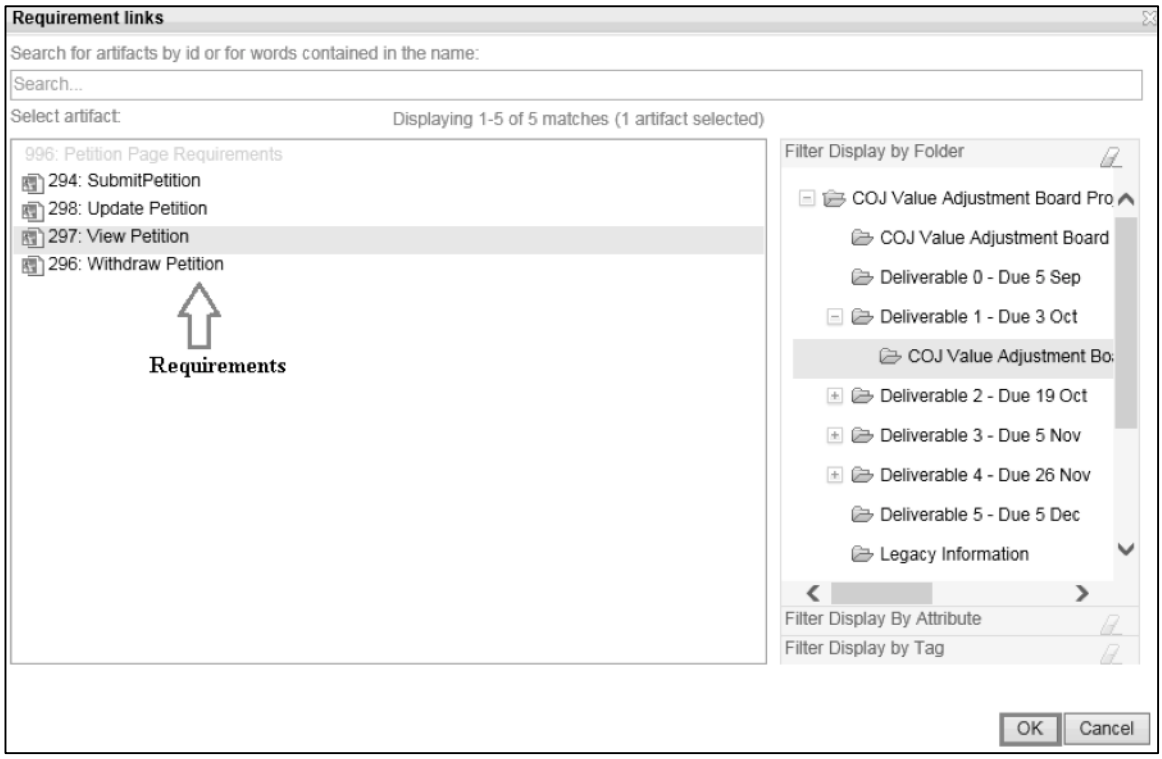


Figure 28. Linking Test Case in Rational Quality Manager to Requirement in Rational Requirements Composer

The linkage between the test case “Verify Petition Information” and “View Petition” can be verified on the “Requirement Links” Section (refer to Figure 29).

Figure 29. Requirement Links Section of "Verify Petition Information" Test Case

These steps were used to create a linkage between a test case and its associated requirement for each of the test cases in the "COJ VAB Project (Quality Management)" project. After linking all the test cases with the requirements, "COJ VAB Project (Quality Management)" in Rational Quality management is linked with "COJ VAB Project (Requirements)" Rational Requirement Management.

4.3 Linking between Project Areas in Rational Team Concert and Rational Quality Manager

After linking the requirements in RRC to test cases in RQM, the last part of completing traceability is linking the test artifacts in Rational Quality Manager to the development artifacts in Rational Team Concert.

This section shows how the link between the COJ VAB Project (Change Management) in RRC and COJ Value Adjustment Board Project (Quality Management) in Rational Quality Manager was created.

1. Log on to the server administration interface of Rational Quality Management Composer server as an Admin group member (user with administrative privileges).
2. Navigate to the project overview page of the "COJ Value Adjustment Board Project (Quality Management)" and click the Add button in the Associations section.
3. Under the Applications select "/qm", under Associations select "Provides – Requirements" and under Artifact Containers select "COJ Value Adjustment Board Project (Quality Management)" (refer to Figure 30).

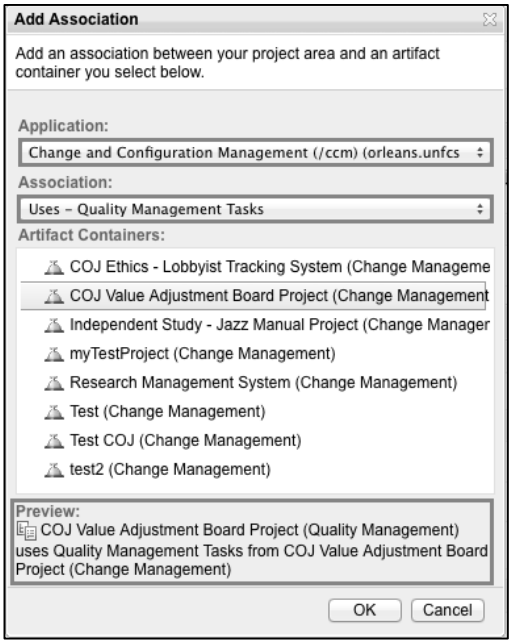


Figure 30. Linking COJ Value Adjustment Board Project (Quality Management) with COJ Value Adjustment Board Project (Change Management)

The linkage between “COJ VAB Project (Quality Management)” and “COJ VAB Project (Change Management)” can be verified by link created in the “uses” section of the project area overview page of "COJ Value Adjustment Board Project" in Rational Requirement Composer (refer to Figure 31).

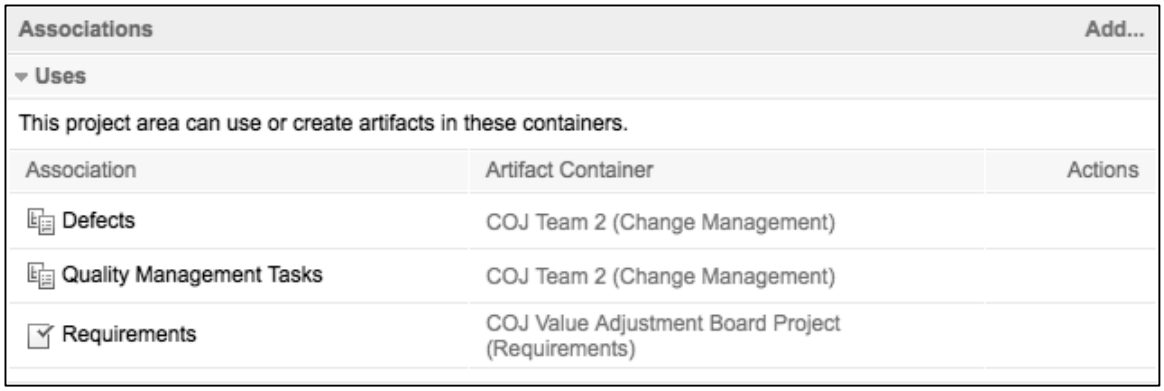


Figure 31. Uses Section of the Project Area Overview Page of "COJ Value Adjustment Board Project"

After establishing project linkage, the next step is to link the artifacts.

A test case in Rational Quality Management and a work item in Rational Team Concert establish the link type between Rational Quality Management and Rational Team Concert.

4.3.1 Linking Test Cases in Rational Quality Manager to Development Work Items in Rational Team Concert

A test case in COJ Value Adjustment Board Project (Quality Management) is linked with a work item in COJ Value Adjustment Board Project (Change Management).

The following steps are used to link a work item with a test case:

1. To link the artifacts, go to the Development Items section of the test case (refer to Figure 32).

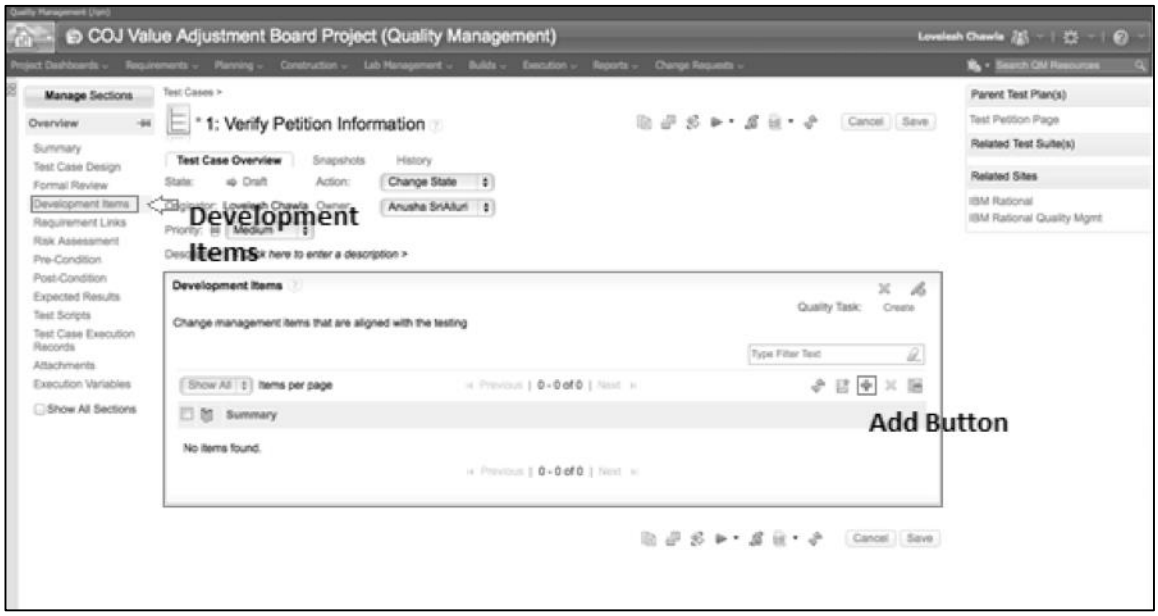


Figure 32. Development Items Section of a Test Case

2. Click the Add button (refer to Figure 32). The Plan Item pop-up will show with "COJ Value Adjustment Board Project (Change Management)" Project area selected by default (refer to Figure 33).

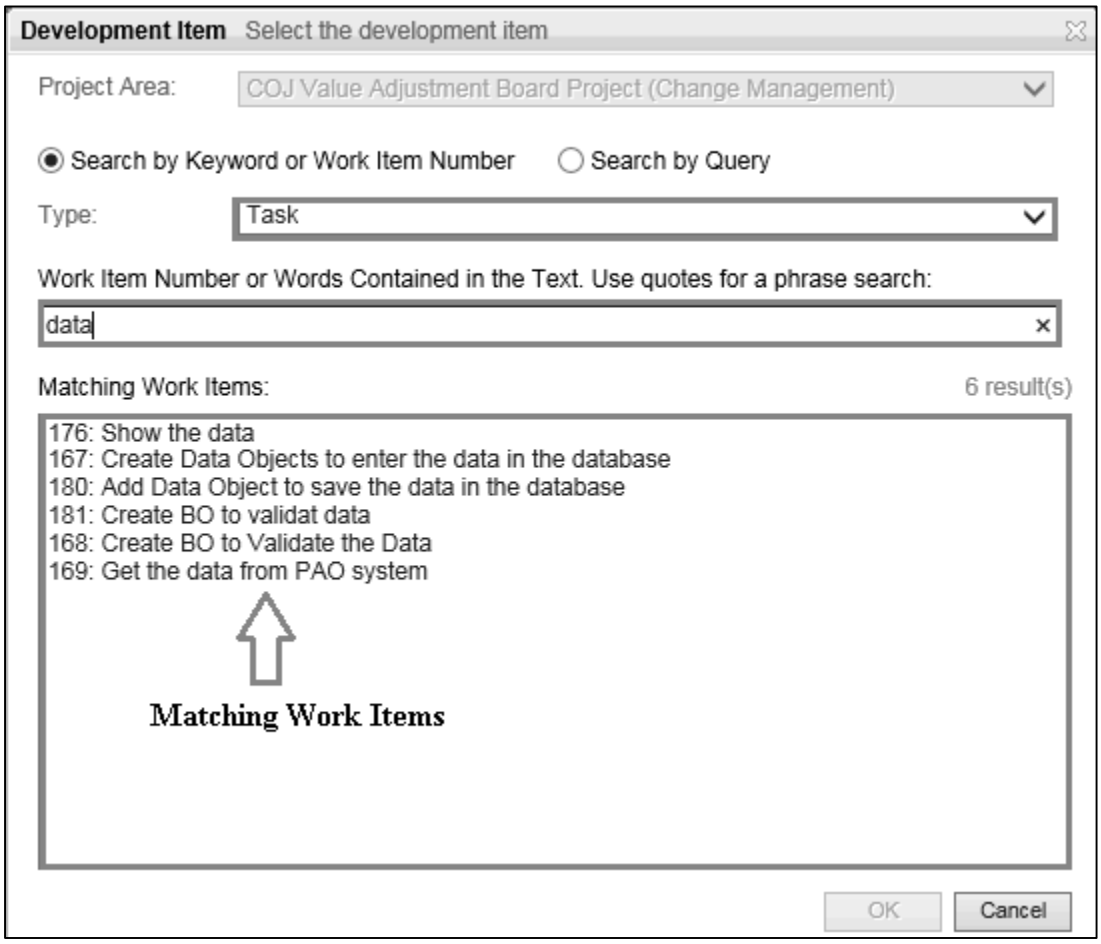


Figure 33. Development Items Selection Pop-up Window

3. Search the work item by choosing type of the work item, and enter the text to search.
4. Choose the work item from the search result, click OK and the selected plan item will be added to the test case.

The chosen work item that is added can be verified in the "Development Items" section of the test case (refer to Figure 34).

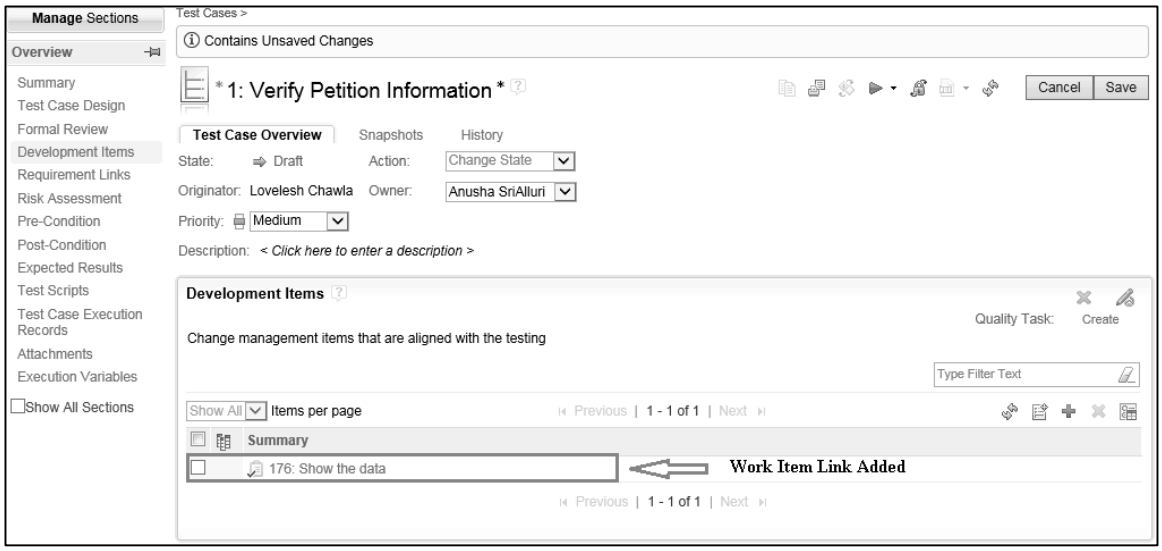


Figure 34. Verification of added Work Item in Development Items Section of "Verify Information" Test Case

These steps were repeated to add the linkage between a test case and associated work items. After linking all the test cases with the work items, "COJ VAB Project (Quality Management)" in Rational Quality management is linked with "COJ VAB Project (Change Management)" Rational Team Concert.

This section completes the artifact linkages that were shown earlier in this section in Figure 10.

Chapter 5

CONCLUSION

5.1 Results

This research presented a case study of traceability in Collaborative Lifecycle Management Framework as applied to the design and development of the Value Adjustment Board Project (VAB) of City of Jacksonville. First-year graduate software engineering students implemented the project using Scrum and Unified Process development methods.

According to Babcock, "CLM is an integrated Application Lifecycle Management solution comprising of three main products: IBM Rational Requirements Composer (RRC), IBM Rational Team Concert (RTC), and IBM Rational Quality Manager (RQM)" [Babcock12]. Therefore, the research methodology proposed the use of RRC to define requirements and RTC to create work items to implement those requirements and create a linkage among them. The thesis also proposed the use of RQM to create test cases to validate the requirements in RRC and form a linkage between the two project areas. Finally, the thesis recommended linkage of work items in RTC to test cases in RQM and establish traceability of the requirements in the VAB project.

The results of the research list detailed steps along with screenshots to link the project area in RRC “COJ Value Adjustment Board Project (Requirements)” with the “COJ Value Adjustment Board Project (Change Management)” project area in RTC. This was followed by creating work items in “COJ Value Adjustment Board Project (Change Management)” for each requirement in the “COJ Value Adjustment Board Project (Requirements)”. Thus the first proposition of research methodology to link RRC to RTC is validated.

Also presented among the results of this research is the approach used to link the project area in RRC “COJ Value Adjustment Board Project (Requirements)” with the “COJ Value Adjustment Board Project (Quality Management)” project area in RQM. The research also showed how to create a test case for each requirement in the COJ Value Adjustment Board Project (Requirements) and hence verify the second proposition of research methodology to link RRC to RQM.

Finally the research methodology presented detailed steps to link the “COJ Value Adjustment Board Project (Quality Management)” with the “COJ Value Adjustment Board Project (Change Management)” and subsequently link the work items and test cases between them. Thus, these results confirm the last guiding proposition defined in the research methodology to link RTC with RQM.

Even though these results confirm all propositions in the research methodology, the project was implemented in an academic environment, which added the following constraints:

- Small scope project –The Software Engineering I course focused on establishing requirements and the Software Engineering II course focused on coding, testing and delivering the software. Since there was only one semester to code, test and deliver the software, the scope of this project was kept small and comprised less than 10 use cases. Huang says that "the number of traceability links to be identified grows exponentially with the size and complexity of the software system" [Cleland-Huang02]. Thus using the CLM framework for larger and more complex projects could not be illustrated.
- Time Constraints – Most of the graduate students had full time jobs and were taking other courses. This limited project time outside the Software Engineering class, which met twice weekly. Our group meetings were generally organized around this course to accommodate the varied and full schedules of our members. But getting team members who frequently work full time was very difficult to manage.
- Learning Curve – Learning IBM's CLM tool alongside software engineering principles and practices slowed down the learning for software engineering students. The CLM tool is indeed very powerful. Unfortunately, with considerable power comes considerable complexity. Thus many of the features of the CLM tool were not used fully and certainly not to capacity.

- Restricted Access – The CLM solution can be accessed by different stakeholders who can view and add artifacts that are produced during the software development cycle. To keep the process simple, the group decided to limit access to the team Project Manager when it came to adding artifacts. Because the Project Manager is not an artifact owner, some of the artifacts or their traceability relationships with other artifacts in retrospect might not have been added. This is uncommon in the software engineering industry and may have contributed to missing traceability links.

The results of this project also reveal some drawbacks of the programs in CLM framework

- Manual Traceability Link Detection – As shown in the research results RRC, RTC and RQM can add traces to link different artifacts among the tools. However, the user has to identify these traceability links between artifacts. The programs in CLM framework do not identify or add traceability links between the artifacts automatically. This can lead to an incomplete traced software project when a user does not identify the link between artifacts.
- Untraced Artifacts – None of the three programs in CLM warns the user of a missed or invalidated requirement if the user did not add a traceability link between a requirement and test case, or a requirement and programming task. This can also lead to an incomplete traced software project.
- Non-intuitive User Interface – The three programs in CLM framework are packed with features. The abundance of these features may make the user interface of

these programs non-intuitive. A user may get lost and may not understand how to trace different artifacts.

- No Query on Trace Links – "Trace links are created to empower trace users to perform various software engineering tasks more effectively. From the technical perspective the user interaction layer must provide tools and infrastructure to leverage available traces, and enable project stakeholders to realize the full benefits of available trace data" [Cleland-Huang14]. None of these tools provides a feature to query and give results of all the traceability links and associated artifacts in a software project. If these results existed they would assist the user with change impact analysis and management, which is a driving factor of traceability.

Despite these drawbacks, results show the IBM CLM tools are user friendly and provide a vast range of relationship options to link requirements to development artifacts and test cases. Results also verify the seamless integration of all three programs (RRC, RTC and RQM) in the CLM framework. Ultimately, these programs provide graphical representations of traces between requirements with programming tasks and between requirements with test cases to ensure all requirements were included and tested in the final deliverable. Thus these programs establish complete traceability of the project and increase the project success rates as shown by Project Resolution Results from CHAOS research (refer to Table 6). Traceability also added benefits in the areas of project management, process visibility, verification and validation, and maintenance [Kannenberg09].

In summary, this research provides significant evidence that the Rational Requirements Composer, Rational Quality Manager, and Rational Team Concert tools in CLM solution may be successfully used to achieve requirements traceability for a small project and ensure project members may easily understand a particular requirement's relation to other aspects of the project. Thus, the research goals of this thesis were successfully achieved and can benefit researchers and practitioners looking for evidence to use the IBM CLM solution to trace artifacts in a small project.

5.2 Future Research

Future research may include the addition of an automated build process to the traceability procedure. Builds may be linked with change sets which include work items completed in the build that trace back to requirements and test cases. These change set links would provide a traceability roadmap for each build.

Additional methods to verify the accuracy and completeness of the mapped traceability links among work items need to be further researched. At the moment, none of the Rational tools can be configured improve their accuracy and completeness of traceability on a project.

REFERENCES

Print Publications:

[Abadi08]
Abadi A., M. Nisenson, and M. Simionovici, "A traceability technique for specifications", Proceedings of the 16th international conference on program comprehension, pp 24-30, 2008

[Alexander02]
Alexander, I., "Towards Automatic Traceability in Industrial Practice", Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2002), Edinburgh, UK, September 2002

[Alexander03]
Alexander, I., "SemiAutomatic Tracing of Requirement Versions to Use Cases – Experience and Challenges", Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003), Canada, October 2003.

[Antoniol02]
Antoniol, G., G. Canfora, G. Casazza, A. De Lucia and E. Merlo, "Recovering Traceability Links between Code and Documentation", IEEE Transactions on Software Engineering, 28(10), 970-983, October 2002.

[Arkley]
Arkley, P., P. Mason and S. Riddle, "Position Paper: Enabling Traceability", Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Scotland, 2002.

[Bianchi00]
Bianchi, A., A. R. Fasolino, G. Vissagio, "An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models" Proceedings of 8th International Workshop on Program Comprehension (IWPC'00), 149-159, Limerick, Ireland, June 2000

[Borg13]
Borg, M., P. Runeson and A. Ardo, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability", Springer Science + Business Media, New York, 2013.

[Capobianco09]
Capobianco G., A. De Lucia, R. Oliveto, A. Panichella and S. Panichella, "Traceability recovery using numerical analysis", Proceedings of the 16th working conference on reverse engineering, pp 195-204, 2009

[Cleland-Huang02]
Cleland-Huang, J., C. Chang and J. Wise, "Supporting Event Based Traceability through High-Level Recognition of Change Events", Proceedings of IEEE COMPSAC Conference, 2002.

[Cleland-Huang03]
Cleland-Huang, J. and D. Schmelzer, "Dynamic Tracing Non-Functional Requirements through Design patter Invariants", Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003), 2003.

[Cleland-Huang10]
Cleland-Huang, J., X. Zou and R. Settimi, "Improving automated requirements trace retrieval: A study of term-based enhancement methods", Empir Software Eng 15(2), pp 119–146, 2010.

[Cleland-Huang14]
Cleland-Huang J., O. Gotel J. Huffman Hayes, P. Mader, and A. Zisman, "Software traceability: Trends and future directions", Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India, 2014.

[Constantopoulos95]
Constantopoulos, P., M. Jarke, Y. Mylopoulos and Y. Vassiliou, "The Software Information Base: A Server for Reuse", VLDB Journal, 1995.

[Dick02]
Dick, J., "Rich Traceability", Proceedings of the 1st International Workshop on Traceability for emerging forms of Software Engineering (TEFSE'02), September, 2002.

[Dogmes98]
Dogmes, R. and K. Pohl, "Adopting Traceability Environments to Project-Specific Needs", Communications of the ACM, 1998.

[Domges08]
Domges, R. and P. Klaus, “Adapting traceability environments to project-specific needs”, Communications of the Acm, 2008.

[Dorfman90]
Dorfman, M., “System and Software Requirements Engineering”, IEEE Computer Society Press Tutorial, 1990.

[Eastbrook98]
Easterbrook, S., J. Callahan and V. Wiels, “V & V through Inconsistency Tracking and Analysis”, Proceedings of the 9th International Workshop on Software Specification and Design, 1998.

[Ecklund96]
Ecklund, E. F., L. M. Delcambre and M. J. Freiling, “Change cases: use cases that identify future requirements”, Proceedings of OOPSLA '96, 1996.

[Egyed02]
Egyed, A. and P. Gruenbacher, “Automatic Requirements Traceability: Beyond the Record and Replay paradigm”, Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE), Edinburgh, United Kingdom, September, 2002.

[Egyed03]
Egyed, A., “A Scenario-Driven Approach to Trace Dependency Analysis”, IEEE Transactions on Software Engineering, Vol. 9, No. 2, February, 2003.

[Fiutem98]
Fiutem, R. and G. Antoniol, “Identifying Design-Code Inconsistencies in Object-Oriented Software: a Case Study”, Proceedings of International Conference on Software Maintenance, Maryland, March, 1998.

[Gethers11]
Gethers M., B. Dit, S. Klock and D. Poshyvanyk, “Traceclipse: an eclipse plug-in for traceability link recovery and management”, Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, pp 24-30, ACM, Waikiki, HI, 2011

[Gotel94]
Gotel, O. and A. Finkelstein, “An Analysis of the Requirements Traceability Problem”, Proceedings of the 1st International Conference in Requirements Engineering, Colorado Springs, 1994.

[Gotel95]
Gotel, O. and A. Finkelstein, "Contribution Structures", Proceedings of 2nd International Symposium on Requirements Engineering, (RE '95), March 27-29, England.

[Hayes03]
Hayes, J. H., A. Dekhtyar and J. Osborne, "Improving Requirements Tracing via Information Retrieval", Proceedings of the 11th IEEE International Requirements Engineering Conference, 2003.

[Hayes07]
Hayes, H., A. Dekhtyar, S. Sundaram, A. Holbrook and S. Vadlamudi, "Requirements Tracing on target (RETRO): improving software maintenance through traceability recovery", 2007.

[IEEE90]
IEEE Computer Society, 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology, 1990.

[Ismenia07]
Ismenia, G. and A. Goknil, "Survey of Traceability Approaches in Model-Driven Engineering", EDOC '07 Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, 2007.

[Jarke01]
Jarke, M. and B. Ramesh, "Towards Reference Models for Requirements Traceability", IEEE Transactions in Software Engineering, 2001.

[Kaindl92]
Kaindl, H., "The Missing Link in Requirements Engineering", Software Engineering Notes, 1992.

[Knethen02]
Knethen, A. V., "Automatic Change Support Based on a Trace Model", Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'02), Edinburgh, September 2002.

[Kozlenkov02]
Kozlenkov, A. and A. Zisman, "Are their Design Specifications Consistent with our Requirements?", Proceedings of IEEE Joint International Requirements Engineering Conference, September 2002.

[Krause03]
Krause, P., G. Spanoudakis, E. Perez-Minana and A. Zisman, "Tracing Software Requirements Artefacts", Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP'03), June 2003.

[Lavazza00]
Lavazza, L. and G. Valetto, "Requirements-based Estimation of Change Costs", Empirical Software Engineering - An International Journal, 5(3), November 2000

[Leffingwell02]
Leffingwell, D. and D. Widrig, The Role of Requirements Traceability in System Development, 2002.

[Leteiler02]
Letelier, P., "A Framework for Requirements Traceability in UML-based Projects", Proceedings of the 1st International Workshop on Traceability for Emerging Forms of Software Engineering, Edinburgh, UK, September 2002.

[Lindval96]
Lindval, M. and K. Sandahl, "Practical Implications of Traceability", Software Practice and Experience, 1996.

[Mader09]
Mäder, P., O. Gotel and I. Philippow, "Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice", Traceability in Emerging Forms of Software Engineering, 2009.

[Maletic03]
Maletic, J.I., E. V. Munson, A. Marcus and T. N. Nguyen, "Using a Hypertext Model for Traceability Link Conformance Analysis", Proceedings of the 2nd International Workshop on Traceability for Emerging Forms of Software Engineering (TEFSE'03), 2003.

[Marcus03]
Marcus, A. and J. I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", Proceedings of 25th International Conference Software Engineering, 2003.

[Mohan02]
Mohan, K. and B. Ramesh, "Managing variability with Traceability in product and Service Families", Proceedings of the 35th Hawaii International Conference on System Sciences, IEEE, 2002.

[Paech02]
Paech, B., A.V. Knethen, F. Kiedaisch and F. Houdek, "Systematic Requirements Recycling through Abstraction and Traceability", Proceedings of the IEEE International Requirements Engineering Conference, Germany, September 2002.

[Parvathy08]
Parvathy A., B. Vasudevan and R. Balakrishnan, "A comparative study of document correlation techniques for traceability analysis", Proceedings of the 10th international conference on enterprise information systems, information systems analysis and specification, pp 64-69, 2008

[Pinheiro96]
Pinheiro, F. and J. Goguen, "An Object-Oriented Tool for Tracing Requirements", Proceedings of the Second International Conference, IEEE, Colorado Springs, 1996.

[Pinheiro00]
Pinheiro, F., "Formal and Informal Aspects of Requirements Tracing", Proceedings of 3rd Workshop on Requirements Engineering, Rio de Janeiro, Brazil, 2000

[Pohl96]
Pohl, K., "Enabling Requirements Pre-Traceability", Proceedings of the 2nd IEEE International, Conference on Requirements Engineering, Colorado Springs, 1996.

[Ramesh92]
Ramesh, B. and V. Dhar, "Supporting Systems Development Using Knowledge Captured During Requirements Engineering", IEEE Transactions in Software Engineering, 1992.

[Ramesh98]
Ramesh, B., "Factors Influencing Requirements Traceability Practice", Communications of the ACM, Vol 41, No 12, 1998.

[Ramesh01]
Ramesh, B. and M. Jarke, "Reference Models for Requirements Traceability", IEEE Transactions, 2001.

[Randell68]
Randell, B., "Software Engineering In 1968", Computing Laboratory University of New Castle upon Tyne, 1968.

[Riebisch01]
Riebisch, M. and I. Philippow, "Evolution of Product Lines Using Traceability", OOPSLA 2001 Workshop on Engineering Complex Object-Oriented Systems for Evolution, Germany, October 2001.

[Rilling07]
Rilling, J., R. Witte and Y. Zhang, "Automatic Traceability Recovery: An Ontological Approach", International Symposium on Grand Challenges in Traceability (GCT'07), Center of Excellence in Traceability, Lexington, Kentucky, USA, March 22-23, 2007.

[Sherba03]
Sherba, S. A., K. M. Anderson and M. Faisal, "A Framework for mapping Traceability Relationships",Proceedings of the 2nd International Workshop on Traceability for Emerging forms of Software Engineering (TEFSE 2003), Montreal, 2003.

[Song98]
Song, X., B. Hasling, G. Mangla and B. Sherman, "Lessons Learned from Building a Web-Based Requirements Tracing System", Proceedings of 3rd International Conference on Requirements Engineering, 1998

[Spanoudakis99]
Spanoudakis, G., A. Finkelstein and D. Till, "Overlaps in Requirements Engineering", Automated Software Engineering Journal, 1999.

[Spanoudakis04]
Spanoudakis, G., A. Zisman, E. Perez-Minana and P. Krause, "Rule-Based Generation of Requirements Traceability Relations", Journal of Systems and Software, 2004.

[Spanoudakis05]
Spanoudakis, G. and A. Zisman, "Software Traceability: A Roadmap", 2005.

[Steven06]
Steven, C. H. and J. Luoma, "Acceptance and Commitment Therapy: Model, Process and outcomes." Behavior Research and Therapy, 2006.

[Strens96]
Strens, M. and R. Sugden, "Change Analysis: A Step towards Meeting the Challenge of Changing Requirements", Proceedings of the IEEE Symposium and Workshop on Engineering of Computer- Based Systems, Fredrichshafen, 1996.

[Sultanov10]
Sultanov H. and H. Hayes, "Application of swarm techniques to requirements engineering: Requirements tracing", Proceedings of the 18th international requirements engineering conference, pp 211-220, 2010
Transactions in Software Engineering, 2001.

[Whitehead97]
Whitehead, E., "An Architectural Model for Application Integration in Open Hypermedia Environments", Proceeding of the 8th ACM Conference on Hypertext, 1-12, April 1997

[Xu02]
Xu, P. and B. Ramesh, "Supporting Workflow management Systems with Traceability", Proceedings of the 35th Hawaii International Conference on System Sciences, IEEE, 2002.

[Zisman02]
Zisman, A., G. Spanoudakis, E. Perez-Minana and P. Krause, "Towards a Traceability Approach for Product Families Requirements", Proceedings of 3rd ICSE Workshop on Software Product Lines, May 2002

[Zisman03]
Cysneiros, G., A. Zisman and G. Spanoudakis, "A Traceability Approach for i* and UML Models", Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems - ICSE 2003, May, 2003.

Electronic Sources:

[Babcock12]
Babcock, C., "IBM Preps Team Concert, A Collaborative Development Tool", http://www.informationweek.com/ibm-preps-team-concert-a-collaborative-d/206905042, 2012, last accessed January 15, 2014.

[Barry86]
Boehm, B. W., "A spiral model of software development and enhancement", http://dl.acm.org/citation.cfm?id=12948, 1986, last accessed January 8, 2014.

[CHAOS13]
The Standish Group, "CHAOS MANIFESTO 2013", http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf, 2013, last accessed March 15, 2015.

[DeRose01]
DeRose, S., E. Maler and D. Orchard, "XML Linking Language (XLink)", http://www.w3.org/TR/2000/REC-xlink-20010627, 2001, last accessed January 15, 2014.

[Gothe08]
Göthe, M., C. Pampina, P. Monson, N. Khurram, K. Patel, B. Smith and N. Yuce, "Collaborative Application Lifecycle Management with IBM Rational Products", http://www.redbooks.ibm.com/abstracts/sg247622.html, December 23, 2008, last accessed January 10, 2014.

[Jazz12]
Jazz Inc, "About Jazz Platform", https://jazz.net/story/about/about-jazz-platform.jsp, 2012, last accessed February 12, 2014.

[Kannenberg09]
Kannenberg A. and H. Saiedian, “Why Software Requirement Traceability Remains a Challenge”, http://www.crosstalkonline.org/storage/issue-archives/2009/200907/200907-Kannenberg.pdf, 2009, last accessed March 15, 2015.

[Peter69]
Naur, P. and B. Randell, “Software Engineering Report on a conference sponsored by the NATO Science Committee”, http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF, 1969, last accessed January 8, 2014.

[RTM14]
Integrated Chipware RTM <Reference>, www.chipware.com, last accessed January 15, 2014

[TELEOLOGIC13]
Teleologic DOORS <Reference>, www.teleologic.com/products/doors, last accessed December 23, 2013

[VAB14]
Value Adjustment Board <Reference>, http://www.coj.net/departments/regulatory-boards-and-commissions/value-adjustment-board.aspx, last accessed January 10, 2014.

[Wood10]
Wood, M., “The Evolution of ALM,” Projectmanagement.com, http://www.projectmanagement.com/articles/255472/The-Evolution-of-ALM, 2010, last accessed January 15, 2014.

VITA

Lovelesh Chawla is currently working as a Senior Software Engineer at CSX in Jacksonville, Florida. He has more than 6 years of experience in IT industry primarily in C# .NET applications. He holds a Bachelor of Sciences degree from Michigan State University, in Computer Science and Engineering and expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida, May 2015.

Lovelesh is fluent in Hindi, English and Punjabi. He is always ready to learn and adapt new tools and technologies. He is originally from India and currently lives with his wife Kelly in Jacksonville, Florida.