Requirements Change Awareness

by

Andrew Swerdlow
BSc, University of Victoria, 2005

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

© Andrew Swerdlow, 2008
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
Supervisory Committee

Requirements Change Awareness

by

Andrew Swerdlow
BSc, University of Victoria, 2005
Abstract

Software development projects are commonly subject to high levels of change, normally due to evolving requirements. Since the requirements are the foundation of software projects and directly linked to the project’s code, tests, and other design components, a lack of awareness of requirement changes has the potential to affect all of the related downstream efforts. As a result, it is essential that all relevant stakeholders in the development process be informed of the changes that influence their work. Improving awareness of changes to requirements has the potential to improve a software developer’s decision-making process as well as improve communication between stakeholders. However, the concept of requirements change awareness has not been well defined in the literature, as it is a new concept. No common language articulates what change awareness information is critical if people are to track and maintain awareness of requirements. Providing a common language would help researchers and practitioners to inform design and evaluate requirements change awareness support tools. This work has two contributions to the field of requirements engineering (RE). First, it contributes a framework that captures the elements important to the concept of awareness in RE. The framework categorizes information about changes to requirements and how those changes affect the rest
of the project. The framework describes different views of awareness combined into a framework intended to capture significant aspects of requirements change awareness. The framework divides requirement change awareness into person-based views, artefact-based views and workspace-based views. The framework examines elements such as the nature of changing requirements and their relationship to project roles. The framework also provides a description of information sources related to requirement changes as well as the group interactions that can spawn change. Second, it contributes insights from the evaluation of three commercial RE management tools: RequisitePro, DOORS, and CaliberRM. Specifically, an evaluation that demonstrated the utility of the framework in evaluating awareness support in RE management tools revealed that the different tools offered a different level of awareness information. While CaliberRM performed better at providing awareness information to users since it had a more complete notion of users and roles that focused on collaboration, RequisitePro gave the least support for requirements change awareness.
# Table of Contents

Supervisory Committee .................................................................................................. ii  
Abstract ................................................................................................................ iii  
Table of Contents ........................................................................................................ v  
List of Tables ........................................................................................................ vii  
List of Figures ........................................................................................................ viii  
Acknowledgements ..................................................................................................... x  

Chapter 1 Introduction .............................................................................................. 1  
1.1.1 Problem Statement ......................................................................................... 2  
1.1.2 Research Goal ............................................................................................... 3  
1.1.3 Research Methodology .................................................................................. 3  
1.1.4 Research Contributions ................................................................................ 4  
1.1.5 Thesis Organization ...................................................................................... 5  

Chapter 2 Background and Related Work ................................................................. 7  
2.1 Requirements Engineering .............................................................................. 7  
2.1.1 Requirements Change Management ......................................................... 10  
2.1.2 Traceability .................................................................................................. 13  
2.1.3 Requirements Management Tools ............................................................. 14  
2.2 Requirements Change Awareness ................................................................. 15  
2.2.1 Awareness Tools ......................................................................................... 16  
2.3 Requirements Engineering and Requirements Change Awareness ........... 19  

Chapter 3 Requirements Change Awareness Framework .................................... 26  
3.1 Framework Construction ............................................................................... 27  
3.1.1 Requirements .............................................................................................. 27  
3.1.2 Artefacts ..................................................................................................... 28  
3.1.3 Software Development Roles ...................................................................... 28  
3.2 Framework Views ........................................................................................... 29  
3.2.1 Person-Based Views ................................................................................... 30  
3.2.2 Artefact-Based View .................................................................................. 41  
3.2.3 Workspace-Based View ............................................................................. 47  
3.3 Information Elements ...................................................................................... 51  
3.4 Summary ......................................................................................................... 54  

Chapter 4 Application of the Framework to the Evaluation of Three Requirements  
Engineering Management Tools ............................................................................ 58  
4.1.1 Tools Evaluated ........................................................................................... 58  
4.1.2 Evaluation Procedure ............................................................................... 58  
4.2 Evaluation Use Cases ...................................................................................... 60  
4.2.1 Problems Encountered during Evaluation ............................................... 73  
4.3 Tool Evaluation: CaliberRM ......................................................................... 74  
4.3.1 Summary .................................................................................................... 99
List of Tables

Table 2-1 Information Elements and Workspace Questions Related to “Where?” ................................................................. 21
Table 2-2 Information Elements and Workspace Questions Related to “Who?” . 22
Table 2-3 Information Elements and Workspace Questions Related to “What?” 22
Table 2-4 Information Elements and Workspace Questions Related to “How?” . 23
Table 2-5 Information Elements and Workspace Questions Related to “When?” 23
Table 2-6 Information Elements and Workspace Questions Related to “Why?” . 24
Table 3-1 Requirements Change Awareness Framework Overview .................. 27
Table 3-2 Framework Views and Information Elements................................ 54
Table 3-3 Framework Views and Information Elements Summary .................. 56
Table 4-1 Use Cases Corresponding to Information Elements....................... 61
Table 4-2 Framework Views and Use Case Coverage .................................. 72
Table 4-3 CaliberRM Use Case Summary ................................................. 100
Table 4-4 RequisitePro Use Case Summary .............................................. 116
Table 4-5 DOORS Use Case Summary ..................................................... 145
List of Figures

Figure 3-1 Affects of Changes to Requirements ................................................................. 33
Figure 3-2 Expected Results: A User Affected by the Changes of Two Distinct Roles ................................................................................................................................. 37
Figure 3-3 Expected Results: Three Effects of User Changes ........................................... 38
Figure 3-4 Interactions between Archetypes in the Workspace ........................................ 48
Figure 4.1 Template Comparison ...................................................................................... 74
Figure 4-2 CaliberRM: Creating Users ............................................................................. 76
Figure 4-3 CaliberRM: Creating Users Role .................................................................... 77
Figure 4-4 CaliberRM: Assigning User and Group Information ....................................... 78
Figure 4-5 CaliberRM: Login .......................................................................................... 79
Figure 4-6 CaliberRM: Adding a Requirement .................................................................. 80
Figure 4-7 CaliberRM: Creating a Requirement ............................................................... 81
Figure 4-8 CaliberRM: Assigning Traceability Links ...................................................... 82
Figure 4-9 CaliberRM: Adding Detail to a Requirement .................................................. 83
Figure 4-10 CaliberRM: Adding Approvals ..................................................................... 84
Figure 4-11 CaliberRM: Assigning a Requirement .......................................................... 85
Figure 4-12 CaliberRM: Attaching Artefacts ................................................................... 86
Figure 4-13 CaliberRM: A Traceability Diagram Plots Linked Requirements ................ 88
Figure 4-14 CaliberRM: A Traceability Matrix ............................................................... 89
Figure 4-15 CaliberRM: A Change Record ...................................................................... 90
Figure 4-16 CaliberRM: Change information ................................................................. 91
Figure 4.17 CaliberRM: Identifying and Deleting Requirements .................................... 92
Figure 4-18 CaliberRM: Checking the Status of Requirements in the Hierarchy ............ 94
Figure 4-19 CaliberRM: Checking the Status of Requirements from the Report .......... 95
Figure 4-20 CaliberRM: Reviewing the Requirements History in the Hierarchy .......... 96
Figure 4-21 CaliberRM: Reviewing the Requirements History Report .......................... 97
Figure 4-22 RequisitePro: Creating a Project ................................................................. 102
Figure 4-23 RequisitePro: Logging in as the Program Manager ..................................... 103
Figure 4-24 RequisitePro: Selecting a New Requirement ................................................ 104
Figure 4.25 RequisitePro: Adding Properties to a Requirement ...................................... 105
Figure 4-26 RequisitePro: Setting the Attributes of a Requirement ............................... 106
Figure 4-27 RequisitePro: Setting Traceability Links ...................................................... 106
Figure 4-28 RequisitePro: Creating Relationships between Requirements .................... 107
Figure 4-29 RequisitePro: Traceability Matrix ............................................................... 109
Figure 4-30 RequisitePro: A Requirement Revision History ......................................... 110
Figure 4-31 RequisitePro: Viewing the Status of a Requirement .................................... 111
Figure 4-32 RequisitePro: Viewing a Requirement-Use Case Matrix ............................ 112
Figure 4-33 RequisitePro: Reviewing Revision Records ................................................ 113
Acknowledgements

I would like to thank Dr. Daniela Damian, my mentor and supervisor. She has inspired me to take on complex problems and never give up; without her support, I would not have been successful in my research.

I would like to thank my committee for providing feedback and for working with my complex schedule.

Thank you, my friend, James Chisan, for providing valuable feedback on my research.

Thank you to Google, who supported me with my research endeavours over the last two years.

Finally, thank you to my parents who have always believed in me and encouraged me to be successful in my academic career.
Chapter 1

Introduction

The primary focus of this thesis is to further our understanding of awareness in requirements engineering (RE). Requirements describe the relationships between changing needs and the objectives of the software development project. They are the foundation of any software project (Brooks, 1987). Successful RE accurately defines the desired characteristics of the intended system (Thayer and Dorfman, 2000). My work rests on the assumption that support for awareness of requirements in software development projects facilitates success. Here, the term requirements change awareness will refer to the concept of managing change information about requirements.

Software development projects are commonly subject to high levels of change, normally due to evolving requirements (Madhavji, 1991). Since the requirements are the foundation of software projects and directly linked to the project code, tests, and other design components, a lack of awareness of requirements changes has the potential to influence all of the related downstream efforts (Sommerville and Kontonya, 1998). As a result, it is essential that all relevant stakeholders in the development process are informed of the changes that affect their work. Improving awareness of changes to requirements has the potential to improve a software developer’s decision-making process as well as communication between stakeholders. However, the concept of requirements change awareness has not been well defined in the literature, as it is a new concept.

While the concept of awareness in software development in general has received some attention (Gutwin and Greenberg, 2001; Dourish and Bellotti, 1996; Damian, Izquierdo, Singer and Kwan, 2007), understanding awareness with
respect to RE has garnered less attention. A lack of awareness of requirement changes can have negative consequences; for instance, a stakeholder without knowledge about changes in project scope can result in incorrectly implemented products or delays in project completion (Sommerville and Sawyer, 1997). A lack of requirement change awareness can also potentially affect the quality of collaboration and communication within teams, since stakeholders will not have a shared understanding about the state of the project requirements (Damian and Zowghi, 2003). However, maintaining awareness of requirement changes is often neglected in practice (Sommerville and Sawyer, 1997).

This research describes the critical information of requirements change awareness in relation to potential affects on software development projects. The results help to develop a framework that facilitates the study of awareness in RE as well as to provide support for practitioners or researchers in evaluating awareness support in requirement management tools. To the best of our knowledge, only a limited literature is available regarding requirement change awareness. Requirement change awareness is not a well-established concept among RE practitioners and the development of a framework enables a logical evaluation of currently available requirement management tools.

1.1.1 Problem Statement

Due to a lack of systematic knowledge of requirement change awareness and to the lack of a framework to allow researchers to explore this concept in a systematic way, we face a corresponding lack of knowledge of what tool features would provide requirement change awareness. Frameworks offer a foundation for researchers to build theoretical models and tools. Frameworks also provide industry practitioners with a way to evaluate tools, to build better tools, and to develop processes that include awareness support. Without the existence of a unified awareness framework, it will be difficult for researchers and practitioners alike to explore jointly concepts related to RE change awareness.
1.1.2 Research Goal

The goal of this research is to develop a framework that captures the important information elements for describing requirements change awareness. The framework highlights the critical information needed if someone is to track information about requirement change awareness and provide researchers a common language for discussing the concepts of requirement change awareness as well as practitioners a mechanism for evaluation of awareness support in requirement management tools.

1.1.3 Research Methodology

This research develops a framework that captures the main concepts of requirement change awareness. Initially, we review the literature on RE and awareness in software development. We then review related work on frameworks and, in particular, identify one framework for asynchronous change awareness in collaborative documents and workspaces (Tam and Greenberg, 2006) that we use as a basis for the development of our framework. We then adapt the framework for use in RE and use the new framework to evaluate awareness support in three state-of-the-art, widely available and used commercial requirement management tools, RequisitePro, DOORS, and CaliberRM. We selected the three tools because they are widely used in multiple industries and have revenues of over 25 million US dollars (Schwaber and Gerush, 2008).

Our literature survey first examines work related to software RE and provides information about change management, traceability, and requirement management tools. We focused the research on those aspects of RE since they are involved with managing changes of requirements. The next step in our literature survey was to review work on awareness in software development. The final step in the literature survey was to review any work that related to requirement change
awareness specifically. While we were not able to find significant research on requirement change awareness specifically, we found a useful framework for discussing asynchronous change awareness in collaborative documents and workspaces by Tam and Greenberg (2006).

This thesis thus utilizes and refines the existing work of Tam and Greenberg (2006) to provide a framework for describing requirements change awareness in software development. They divide their framework into six sections: Where, Who, What, How, When, and Why. Each section is concerned with the affect of change on the information elements of a project. We examine these concepts in Section 2.3. In Chapter 4, we adapt Tam and Greenberg’s (2006) framework approach to focus more specifically on requirements awareness rather than just shared documents and workspaces. The changes to the framework involve changes to the questions posed in the framework construction. Our approach was to ask questions that are more specific to RE; for example, when examining artefact-based views of RE, we ask questions like, “Which artefacts were affected by changed requirements?”

Following the development of the framework, we evaluated three state-of-the-art RE tools: RequisitePro, DOORS, and CaliberRM. The tools were assessed for awareness support in the context of our framework. The aim was to apply the framework to evaluate the awareness capabilities of RE tools and to obtain experience and insights that would allow us to improve the framework.

1.1.4 Research Contributions

This work has two contributions to RE. First, it contributes an RE awareness framework that examines the elements important to the concept of awareness in RE. The framework articulates what change awareness information is critical if people are to track and maintain change awareness of requirements. The framework divides requirements change awareness into person-based views, artefact-based views and workspace-based views, where each different view
offers information focused on each perspective. The framework examines elements such as the nature of changing requirements and their relationship to project roles. The framework also grants a description of information sources related to requirements changes as well as the group interactions that can spawn change. A practical application of the framework is support in the design of awareness features into requirements tools, and evaluation of change awareness features in existing requirements management tools.

Second, the work contributes insights from the evaluation of three commercial RE management tools, RequisitePro, DOORS, and CaliberRM. We found that each tool provided a different level of awareness information. Specifically, CaliberRM performed better at providing awareness information to users, while RequisitePro had a minimal concept of users and roles and offered little requirements change awareness support.

1.1.5 Thesis Organization

Chapter Two provides a literature review of material related to RE processes and practices. The chapter also focuses on reviewing the literature related to providing awareness within collaborative software development projects. The chapter then reviews the limited literature related to the study of awareness in the context of RE processes.

Chapter Three is dedicated to developing a framework to describe requirements change awareness. The framework describes different views of the important components relating to requirements change awareness and their corresponding information elements within the project environment.

Chapter Four begins by presenting data from three different evaluations of commercial RE management tools. The chapter includes a matrix comparison of the tools and their level of support for each element of the framework described in Chapter Three.
Chapter Five discusses the results of the evaluations in Chapter Four. This chapter presents a comparison of the evaluated tools within the context of our framework. The chapter concludes by offering suggestions to improve the framework. In concluding, Chapter Six provides a summary of the work thus far conducted as well as an indication of work for the future.
Chapter 2

Background and Related Work

The first section of this chapter reviews software requirements engineering (RE) literature and provides information about change management, traceability, and requirements management tools. Section 2.2 describes requirements change awareness tools. Finally, Section 2.3 describes requirements change awareness and RE, providing an overview of how they are related.

2.1 Requirements Engineering

Requirements, the documented needs of a proposed system, are the foundation for any software project. Software RE is the science and discipline concerned with establishing and documenting software requirements (Thayer and Dorfman, 2000). Most downstream software development activities such as design, coding, and testing use information about requirements. Research indicates the importance of software RE for successfully developing software products. For example, Brooks remarks that, “no other part of the work [of software development] so cripples the resulting system if done wrong.” Indeed, many studies enumerate the importance of good RE practices. Boehm (1981) provides a classical example of how important requirements are from a business standpoint. Boehm shows that the cost of changing requirements grows exponentially over a project’s lifecycle, stating, “if it costs one dollar to change a requirement in the early planning stages of the software project it can cost 200 dollars when the project is in the final testing stages.”

Some software development methodologies, such as the Waterfall development model (e.g., Royce 1970), imply that RE is a phase performed only
at the start of the software development lifecycle. However, good requirements practices do not merely start and end at the beginning of a project; they are an important part of a product’s entire lifecycle. In fact, Davis (1993) proposes that requirements have the potential to affect every stage of software development. For example, modifying requirements based on initial deployment feedback might require changing the implementation of certain features. Contrary to Davis’ observations, in reality (Sommerville and Sawyer, 1997), developers often neglect requirements in the latter stages of the software development lifecycle. Accordingly, to provide successful RE and management practices, it is essential that requirements are maintained throughout the project lifecycle.

Many reasons explain why requirements are neglected in practice; in sum, it can be difficult and costly to capture all requirements. Many software projects have unique challenges specific to their development environment, such as technological constraints, organizational culture, and project complexity. Requirements need collection and management in a manner that suits the unique environment of the project. This makes providing a single one-size-fits-all approach to RE a challenging task. Nonetheless, some have proposed best practices and processes for practicing effective RE in software development. For example, Sommerville (1998) offers a good foundational overview of requirements processes as well as providing details on requirements classifications, such as functional and non-functional requirements.

Research shows that improved requirements practices lead to an increase in the efficiency of development processes. In their research, Damian and Chisan (2006) provide data that suggest improved requirements engineering processes can positively affect productivity, software quality, and risk management. Their research involved a 30-month case study with ACUS (Australian Centre for Unisys Software), a large software manufacturer in Australia, which was undergoing a requirements engineering processes improvement exercise. ACUS implemented techniques such as feature decomposition, requirements traceability,
group analysis sessions, cross-functional team reviews, structured requirements specifications, and requirements validation testing. The case study demonstrated that ACUS benefited immensely from adopting formal techniques and that the benefits were felt throughout the project lifecycle.

Sawyer, Sommerville, and Viller (1999) propose using tools, such as checklists and templates, to improve requirements practices and increase the quality of software projects. They describe an incremental approach to improving requirements processes based on the Capability Maturity Model for Software. Some of the best practices mentioned are requirements identification, requirements management, traceability policies, and change management. Sawyer et al. indicate that following these best practices can improve productivity, product quality, and time to market.

Good requirements practices can also reduce the risk of project failure and cost overrun. Brodman and Johnson (1995) demonstrate that requirements processes can allow developers to manufacture better software and minimize the inherent risk of large projects. They surveyed 35 organizations about their software processes and found that more mature organizations spend time tracking the requirements that helped to stabilize projects. This stability adds to the overall improvement of the software development process, yielding organizations with quantifiable benefits. Alternatively, incomplete and unstable requirements lead to risky, unpredictable projects. As projects grow, requirements can become more difficult to maintain (Brooks, 1975). Since highly complex projects require considerable resources and coordination, good RE practices can aid in managing the complexity of ever-growing software projects.

Faulk (1997) mentions that many of the errors related to requirements are consequences of issues such as a lack of understanding and communication between stakeholders, and the ineffective management of changing requirements. All of these issues result from lack of requirements change awareness within the workspace (Chisan, 2005).
2.1.1 Requirements Change Management

Haker, Eason, and Dobson (1993) discuss the need to understand and manage changing requirements. To do so, Haker et al. classify requirements as mutable, emergent, consequential, adaptive, or migratory. Any particular classification is subject to the circumstances under which requirements change throughout the project lifecycle. Mutable requirements result from dynamic changes in the markets that the business occupies; these changes normally result from external factors unrelated to organizational changes. Emergent requirements are the effects of the evolution of a project; as the understanding of a project increases, new requirements are added to help clarify project specifications. Consequential requirements come to light after a system has been used or a prototype demonstrated. These requirements are usually introduced via a user request to upgrade the system to meet changing usage patterns. Adaptive requirements are those resultant from the personalization or customization of a system. Finally, migration requirements refer to those requirements needed at intermediate stages of the development cycle. Most commonly, they describe the needs of a project when incrementally moved from one state to another. Classifying requirements in this way allows better understanding of the types of change that can affect software development projects.

Jones (1996) suggests another important factor in understanding why requirements change. He indicates that development of new technologies for which domain knowledge has not matured, such as a new type of navigation system, may demand changes to requirements. Lack of experience with a platform or technology is a large contributor to requirement change. That software tries to automate tasks hitherto non-automated, Jones posits, indicates an intrinsic lack of understanding of the necessary requirements to solve a given problem.
Besides changes to product specifications and environment, other causes for modifications to requirements may lie with miscommunication, changing financial considerations, and other organizational variables. In fact, Madhavji (1991) provides an interesting model of change in the software development environment. Intended to demonstrate an ordered approach to handling ongoing changes to a project, the model defines requirement dependencies and requires gathering feedback, performing outcome assessments, making future projections, and recording metadata.

Outcome analysis is a vital objective of the model proposed by Madhavji. Upon identification of the root cause of change, information about the modified requirement needs communication to all relevant stakeholders. Indeed, Faulk (1997) states that errors often result from failure to communicate requirement changes to other stakeholders. For example, consider a developer implementing an early requirement in the development of a database of items and their prices. If this requirement changes, without proper management, late in the development cycle to include a description attribute, the product’s testing cycle will fail to test the database fully for all required functionality. The mismanagement of the change to the requirements will undoubtedly introduce errors into the system.

Lack of communication flow to downstream stages of software development about requirements change can, as demonstrated, introduce unnecessary instability to the project. Boehm (1991) confirms that when requirements change, stakeholders face higher levels of risk. Boehm suggests several techniques for limiting this exposure. For instance, instituting high thresholds for change or abstracting information and deferring change to different stages of development can mitigate instability. Boehm advocates the simplification of the development process by the limitation of the normal causes of change. Jones (1996) provides case studies in which poorly communicated midstream requirements change led to project failure. He suggests preventing expanding requirements by quantifying the possible affects on other requirements.
and renegotiating requirements based on the potential for change. In general, these proposed best practices attempt to minimize the amount of change during the development process or, alternatively, postpone change for the next iteration of the project.

Many different methods manage changes to requirements. Some of the more notable techniques detailed by Jones (1996) are prototyping, requirements inspections, and joint application development. Prototyping involves the development of mock-ups that represent system functionality or attributes without implementing the entire system. The purpose of such prototypes is to aid in the formative evaluation of a system’s development. Davis (1995) provides two different classifications of prototypes in software development: throwaway and evolutionary. Throwaway prototyping refers to the quick development of models discarded later. Evolutionary prototypes are models of a system that are refined and expanded over time.

Requirements inspections are a way of reviewing software development projects. Ackerman, Buchwald, and Lewsky (1989) offer a good overview on performing software inspections with the hope of finding defects in the systems. They indicate that a set of essential delimiters for performing requirements inspections include frequent inspection by at least three peers, the participation of the producer of change, and the production of consistent data. The intent of the inspection should be the identification of all issues and defects with the software product.

Joint application development is a technique that involves users in the development processes. Reported to provide a greater degree of completeness to the final product, it has the potential to remove 50% of the defects in the requirements phase of development (Carmel, Whitaker, and George, 1993).
2.1.2 Traceability

Traceability is another method for managing change. Requirements traceability is the ability to understand contextual information about a specific requirement throughout the software development lifecycle. Gotel and Finkelstein (1994) describe requirements traceability as “the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).” Gotel and Finkelstein argue that, in practice, it is difficult to maintain traceability, and that many issues related to RE follow from poor requirements traceability practices. To alleviate some of these issues, they propose breaking requirements traceability into pre-requirements specification traceability and post-requirements specification traceability. Pre-requirements traceability refers to linking requirements to originating declarations; whereas post-requirements traceability refers to linking requirements back to the project requirements specification baseline.

Ramesh and Jarke (1998) describe the results of a large-scale study on requirements traceability. This case study identifies several classifications of traceability, which aid in describing the definition of relationships between requirements. These are satisfaction links, evolution links, rationale links, and dependency links.

Satisfaction links describe the connection between requirements and completed components; they demonstrate the fulfillment of requirements by the system. These links also ensure that all components realize some requirements of the system – that is, they limit redundancy in requirements. Evolution links exist between requirements that change from one state to another. On modification or replacement of a requirement, a link between the two states shows the evolution from one requirement to another. This is important since it can improve user awareness of how a requirement changed and supply some background
information or rationale for the evolution. Rationale links identify relationships between requirements and the motivation behind their conception or revision. This provides useful background information related to requirements. Dependency links show relationships between requirements that have some reliance on each other, such as subsystems that require specific interactions. This information helps a developer understand how requirements correlate and to highlight causal relationships. This is also important when assessing the affects of changing requirements.

In summary, requirements traceability grants to stakeholders valuable information on the status of a project. Most importantly, stakeholders can use the process to map requirements change events in a project lifecycle and better understand the affects of these changes (Hamilton and Beeby, 1991).

2.1.3 Requirements Management Tools

Several commercial tools available for managing requirements are gaining popularity with industry practitioners. Rational RequisitePro from IBM uses traceability views (which are detailed visualizations of the relationships between project requirements) to help manage requirements and use cases; Telelogic’s DOORS supports recording, structuring, managing, and analyzing requirements; and, finally, Borland’s CaliberRM offers recording, storing, and validating requirements, as well as the ability to trace requirements throughout a project lifetime. The primary objectives of all three tools are to manage requirements while increasing collaboration and communication in software development environments. Although developed as requirements management systems, these tools also provide some level of awareness support for requirements in software development projects. The tools will be discussed in more detail in Chapter 4.

This thesis will focus on examining state-of-the-art stable commercial tools in Chapter 4. It should be noted that some open source tools, as well, such as
the Open Source Requirements Management Tool, exist. However, this tool is still a work in progress (Sourceforge.net, 2008) and we will not evaluate it or other less widely used tools. In our experience the three main tools used by practitioners are RequisitePro, DOORS, and CaliberRM.

2.2 Requirements Change Awareness

In a general context, following Schmidt (2002), we define user awareness as the understanding of the user’s environment. Similarly, when referring to software development, Gutwin and Greenberg (2001) describe awareness as having comprehension of and knowledge about one’s environment as it changes over time, noting that participation in the changing environment forms awareness. Furthermore, Dourish and Bellotti (1996) characterize awareness as one’s comprehension of the actions of others in the local environment: information acquired in this way provides perspective for one’s own actions. The research set forth here focuses on requirements change awareness by examining the information sources and activities related to RE processes.

Software development is a highly complex activity that normally requires groups to interact and communicate frequently to accomplish tasks (McGrath and Hollingshead, 1994). Whitehead (2007) reports that coordination and collaboration help teams handle continuous change in software development. In particular, Whitehead identifies different group-collaboration tools used to support collaboration, such as a software configuration management tool, which allow users to coordinate recording of change information and to reduce change dependencies between developers.

Awareness is important to team communication because collaboration requires user consciousness of the actions of other users in the project (Endsley, 1995). On the importance of awareness to communication in software development, Hupfer, Cheng, Ross, and Patterson (2004) performed studies analyzing the ways in which people collaborate in development environments
and, in turn, how awareness is affected. This research involved users collaborating with a specialized tool called Jazz, which supported their collaboration and increased awareness.

Lack of awareness can also lead to rework and coordination problems, as described by Damian, Izquierdo, Singer, and Kwan 2007. These investigators conducted a four-month case study at IBM where they examined the development practices of distributed software development teams. They found that awareness was negatively affected by such factors as conflicting organizational cultures and information overload. This lack of awareness in the distributed teams caused issues with communicating change information throughout the team. Their study indicated that awareness information was vital in coordinating interdependent tasks between the developers.

### 2.2.1 Awareness Tools

Indeed, the last decade has seen several attempts to create tools to provide awareness in software development projects. In particular, interest has grown in adding awareness functionality to existing development tools, such as Ariadne and EGRET, both of which are development environments that include additional awareness support (Trainer, Quirk, de Souza, and Redmiles, 2005; Sinha, Sengupta, and Chandra, 2006). In both cases, these tools were developed primarily for academic research and have not seen wide adoption in industry. Ariadne is a Java plug-in for Eclipse intended to offer authorship information about the Java projects and to identify program dependencies. It builds social network graphs by examining the configuration management repository. Ariadne never received wide acceptance in commercial projects and its use remains limited to provision of information based on Java code. EGRET (Eclipse-based Global Requirements Tool) was developed as a prototype with the intent to make available support for distributed requirements management (Sinha, Sengupta, and
Chandra, 2006). It provides users with several features, such as the Artifact Explorer, a view of project requirements in a hierarchical fashion. The tool also presents to users a collaborative component that allows project members to e-mail each other and discuss requirements. *EGRET* also offers traceability links allowing users to view the relationships between requirements.

Other tools attempt to solve the awareness problem. The *Hipikat* project aids new developers to become more familiar with new projects by incorporating an awareness mechanism into the users’ integrated development environment (Cubranic and Murphy, 2003). *Hipikat* promotes awareness by recommending artefacts, such as *Bugzilla* tracking tickets and communication records, concurrent versioning system repositories for source code, and context-sensitive online documentation for developers.

Mockus and Herbsleb (2002), who proposed a tool called the Expertise Browser, suggested another approach for increasing awareness. Their tool allows users to identify experts quickly and easily, optimizes staffing resources, and provides alternatives when experts are not available. Additionally, other projects have approached the issues of awareness. *Workspace Navigator*, described by Ju, Ionescu, Neeley, and Winograd (2004), promotes awareness within a physical work environment. Some of the challenges Ju et al. describe are the delivery of an appropriately detailed amount of information to a user and the avoidance of irrelevant information. Sarma, Noroozi, and Van der Hoek (2003) describe a method of pushing the flow of information out to developers, which is useful when developing customized awareness tools. Cadiz Fussell, Kraut, Lerch, and Scherlis (2002) describe the asynchronous presentation of awareness information. Other investigators have studied how current projects acquire their awareness information using general purpose tools such as e-mail and IRC (Internet Relay Chat) (Gutwin, Penner, and Schneider, 2004). Gutwin et al. discuss how such simple text-based tools can convey a high level of awareness information.
Nonetheless, to our knowledge none of the awareness tools specifically addresses requirements change. This is likely because little information is available regarding the adequate provision of requirements change awareness information. Storey, Cubranic, and Germán (2005) document some of the tools that offer general visual awareness support for software development environments. These investigators surveyed twelve different tools and examined the level of visual awareness support they provide for general software development activities. They evaluate each tool framework in terms of five separate dimensions: intent, information, presentation, interaction, and effectiveness. The intent dimension examines the motivation for providing visual awareness support. The information dimension examines the data each tool uses to build awareness visualizations. The presentation dimension helps categorize how the tools deliver awareness information to users. The interaction dimension aids in determining how users will interface with the awareness information. Finally, the effectiveness dimension evaluates the feasibility of the tools and the extensiveness of their use.

While the framework presented is useful for the comparison of different awareness tools and, indeed, provides a foundation for the research of this thesis, Storey et al. (2005) conclude that such awareness tools, nevertheless, remain unused in industry practice. This thesis begins with the speculation that the reason for the low success rate of existing tools is that tool developers have not had a complete understanding of the concept of awareness. While some, such as Schmidt (1998, 2002), have attempted to provide a high-level overview of awareness and enumerate some of the challenges in providing awareness information, yet a great deal of confusion and debate still occurs over what awareness means in software development environments (Schmidt, 1998).
2.3 Requirements Engineering and Requirements Change Awareness

In traditional software development projects--that is to say, projects without specific awareness support--research has indicated that the dissemination of requirements change to all stakeholders is a lengthy process (Catledge and Potts, 1996). This delay in provision of change information can cause errors in software implementations. In fact, changes to requirements can have ripple effects in a project (Madhavji, 1991) where one event sets off other events in unexpected ways. The effects of a change can be difficult to track without the aid of specialized tools to assess affects to the software development project. It is, therefore, important to understand the scope of changes to requirements in a software project (O’Neal and Carver, 2001).

Research has been conducted on how to manage, change, and provide awareness in software environments (Nuseibeh, 2001). Unfortunately, it is difficult to ascertain a single awareness solution that will work for all environments and projects. As Nuseibeh demonstrates, ignoring change can cause many issues in software projects. Ultimately, if change is not managed the resulting product will not meet the needs of all involved stakeholders and, hence, will fail. Furthermore, Herbsleb and Grinter (1999) indicate that the distribution or dispersion of team members will negatively affect awareness, so managing change and providing awareness support in distributed environments is of particular importance.

While activities that increase awareness of project changes help to avoid issues, providing awareness can be a complex undertaking (Cadiz, Venolia, Jancke, and Gupta 2002). It is important when providing awareness to gather information about changes to the project and deliver it to the appropriate individuals (Kobyliński, Creighton, Dutoit, and Bruegge, 2002). Unfortunately, it is difficult to design awareness tools that accomplish this, yet which do not
distract or annoy users (Espinosa, Cadiz, Rico-Gutierrez, Kraut, Scherlis, and Lautenbacher, 2000).

Some significant components of developing a good awareness tool have already been established: tools should be non-intrusive, scalable, flexible, and configurable (Gutwin, Penner, and Schneider 2004). It is also important to consider how to deliver awareness data to users. Steinfield, Jang, and Pfaff (1999) describe examples of data delivery approaches, such as passive or active delivery of awareness information. In the passive mode, information is delivered without requiring any specific actions of the stakeholders. In active acquisition, the stakeholders must specifically request the information.

Providing a more complete framework of awareness has the potential to help developers become more productive by augmenting their capabilities to assimilate relevant change information. Such a framework should inform specialized awareness tools, which would allow developers to increase their overall understanding of a project, improving the efficiency and accuracy of their work product.

It is the purpose of this thesis to bring together these various existing ideas into one complete framework for describing awareness. Although one can find methods for describing awareness (e.g., Gutwin, Penner, and Schneider 2004), very few focus on RE. Moreover, many models suffer from information overload, that is, they try to do too much, which inevitably becomes counterproductive.

As a basis for the development of a framework for requirements change awareness, this research uses the framework introduced by Tam and Greenberg (2006) for providing asynchronous awareness in collaborative documents. Their paper is important to our research since it provides a methodology for creating a framework for awareness tools; however, their framework is lacking in several areas discussed later in this thesis. Their framework centres on six questions: Where? Who? What? How? When? and Why? Each question is concerned with the affect of change on the information elements of a project. Information
elements are the pieces of knowledge acquired by people that support awareness. They view the affect of change of these information elements from three perspectives: Artefact, Person, or Workspace.

Tam and Greenberg (2006) begin to construct their framework by examining questions related to where information elements affect awareness. As Table 2-1 indicates, they focus their investigation by developing questions related to elements such as location history, gaze history, and edit history from the context of artefacts, people, and workspaces. For example, when examining the information element location history in the context of the artefact-based view, questions such as, “Where was the artefact?” are generated. When examining the information element in the context of a person-based view, questions such as, “Where in the workspace has the person visited?” are generated. From the perspective of the workspace, the investigator may produce questions such as, “Where have people been in the workspace?”

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “Where?”</th>
<th>Artefact-based view</th>
<th>Person-based view</th>
<th>Workspace-based view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location history</td>
<td>Where was this artefact (when I left)?</td>
<td>Where in the workspace has a person visited?</td>
<td>Where have people been in the workspace? Where were artefacts in the workspace?</td>
<td></td>
</tr>
<tr>
<td>Gaze history</td>
<td>Where is the artefact now?</td>
<td>Where in the workspace has a person looked?</td>
<td>At which parts of the workspace have people looked?</td>
<td></td>
</tr>
<tr>
<td>Edit history</td>
<td>Where has this artefact been during the time that I have been away?</td>
<td>Where in the workspace has a person made changes?</td>
<td>In which parts of the workspace have people made changes?</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1 Information Elements and Workspace Questions Related to “Where?”
Source: Tam and Greenberg, 2006

The next dimension of their framework focuses on questions related to Who? They identify the information elements presence history, identity, readership history, and authorship history. Presence history is concerned with
tracking people in the workspace. Identity is established to provide information about the specific stakeholders in the project. Readership and authorship history refer to information about which stakeholders have read and written documents. The questions develop from the perspective of artefacts, people, and workspaces. Table 2-2 outlines questions related to Who?

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “Who?”</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Artefact-based view</td>
</tr>
<tr>
<td>Presence history</td>
<td>Who has looked at this artefact?</td>
</tr>
<tr>
<td>Identity</td>
<td>Who has changed this artefact?</td>
</tr>
<tr>
<td>Readership history</td>
<td></td>
</tr>
<tr>
<td>Authorship history</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-2 Information Elements and Workspace Questions Related to “Who?”
Source: Tam and Greenberg, 2006

Tam and Greenburg (2006) then define the What? dimension of their framework. The sole information element of this dimension deals with action history and details information related to the activities that users perform in the workspace. Table 2-3 indicates that the questions develop from the perspective of artefacts, people, and workspaces.

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “What?”</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Artefact-based view</td>
</tr>
<tr>
<td>Action history</td>
<td>What changes have been made to the artefact?</td>
</tr>
<tr>
<td></td>
<td>What artefacts have a person changed?</td>
</tr>
</tbody>
</table>

Table 2-3 Information Elements and Workspace Questions Related to “What?”
Source: Tam and Greenberg, 2006
They then define the next dimension of their framework, focusing on questions related to How? The information elements that they identify are *process history* and *outcome history*. Outcome history captures the changes in state from the beginning state to the final state. Process history provides incremental information on how the project has changed. As expected, the questions are framed in the context of artefacts, people, and workspaces. Table 2-4 summarizes the questions.

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “How?”</th>
<th>Artefact-based view</th>
<th>Person-based view</th>
<th>Workspace-based view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process history</td>
<td>How has this artefact changed?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outcome history</td>
<td>How has a person changed things?</td>
<td></td>
<td></td>
<td>How has the workspace changed?</td>
</tr>
</tbody>
</table>

**Table 2-4 Information Elements and Workspace Questions Related to “How?”**

Source: Tam and Greenberg, 2006

The next dimension of their framework focuses on questions related to When? The sole information element they focus on is *event history*. The event history provides a temporal view of changes within the workspace. The questions arise from the perspective of artefacts, people, and workspaces.

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “When?”</th>
<th>Artefact-based view</th>
<th>Person-based view</th>
<th>Workspace-based view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event history</td>
<td>When was this artefact changed?</td>
<td></td>
<td></td>
<td>When were changes made to the workspace?</td>
</tr>
<tr>
<td></td>
<td>When was a particular change made to this artefact?</td>
<td></td>
<td></td>
<td>When did a particular change occur in the workspace?</td>
</tr>
<tr>
<td></td>
<td>In what order were changes made to this artefact?</td>
<td></td>
<td></td>
<td>In what order did changes occur to the workspace?</td>
</tr>
<tr>
<td></td>
<td>When did a person make changes?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When did a person make a particular change?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>In what order did this person make changes?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2-5 Information Elements and Workspace Questions Related to “When?”**

Source: Tam and Greenberg, 2006
The final dimension of their framework focuses on questions related to Why? The information elements that they identify are *cognitive history* and *motivational history*. Cognitive history provides information about the underlying logic of changes in the project. Motivational history supplies information about the emotional reasons for making changes. These two elements are very similar in that they both elicit why users made changes. Tam and Greenberg separate them because they help discern accidental, negative changes. The questions, as shown in Table 2-6, develop from the perspective of artefacts, people, and workspaces.

<table>
<thead>
<tr>
<th>Information elements</th>
<th>Specific questions for “Why?”</th>
<th>Artefact-based view</th>
<th>Person-based view</th>
<th>Workspace-based view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cognitive history</td>
<td>Why was this artefact changed?</td>
<td></td>
<td>Why did a person make that change?</td>
<td>Why was that change made in the workspace?</td>
</tr>
<tr>
<td>Motivational history</td>
<td>Table 2-6 Information Elements and Workspace Questions Related to “Why?”</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Tam and Greenberg, 2006

The authors use the framework to generate a tool evaluation check list. This check list provides an overview of each information element and an indication of the level of support each tool offers for each element. Tam and Greenberg use this framework to evaluate an awareness tool called *PastDraw*. Through that evaluation, the authors conclude that the tool needed improvement in several areas if it were to generate comprehensive awareness.

Through their work, Tam and Greenberg (2006) present a reasonable methodology for creating a framework for awareness tools; however, it does not apply well to RE tools. In RE, other types of information may generate awareness information about changing requirements. For instance, information related to roles and traceability is indispensable. Information about roles in software development is very important because roles carry implicit information regarding priority and expertise information. The concept of roles will be examined in detail in Chapter 3. As well, traceability can carry information about relationships
between people, requirements, and artefacts. These traceability relationships can identify how changes will affect the rest of the project, thereby increasing awareness. Nonetheless, this thesis takes the work of Tam and Greenberg (2006) as a starting point and refines it to generate a framework for describing requirements change awareness in software development.
Chapter 3

Requirements Change Awareness Framework

The purpose of our framework is to articulate what change awareness information is critical if people are to track and maintain change awareness of requirements. The framework presents a unified view of the interplay between requirements and awareness. This provides a rationale-based approach for designing systems that can increase levels of awareness and for evaluating existing tools.

Here, we define a structure for organizing and developing our concept of requirements change awareness. In Table 3-1, we present a preview of the requirements change awareness framework. The table represents how our requirements change awareness framework utilizes the work of Tam and Greenberg (2006) and adapts it to reflect specific awareness needs related to traceability and roles. While we have adapted a number of information elements from Tam and Greenberg’s framework, some are new in our framework. The latter are explained in detail in this chapter. Specifically, Section 3.2 will describe the important questions when examining requirements change awareness and Section 3.3 will provide a mapping of the questions to the information elements.
### Information Elements

<table>
<thead>
<tr>
<th>Information Elements</th>
<th>Questions about Awareness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical location history</td>
<td>Where in the world did the user make the change to requirements?</td>
</tr>
<tr>
<td>Change history</td>
<td>What change to requirements did the user make?</td>
</tr>
<tr>
<td>Workspace location history</td>
<td>Where in the workspace did the change to requirements happen?</td>
</tr>
<tr>
<td>Change affect history</td>
<td>What requirements or users were affected by the change to requirements?</td>
</tr>
<tr>
<td>Interaction history</td>
<td>What interactions spawned the change to requirements?</td>
</tr>
<tr>
<td>Authorship history</td>
<td>Who made the change to requirements?</td>
</tr>
<tr>
<td>Outcome history</td>
<td>What is the current status of the requirement?</td>
</tr>
<tr>
<td>Process history</td>
<td>Incrementally, how did the requirement evolve from its previous state?</td>
</tr>
<tr>
<td>Event history</td>
<td>When was the change to the requirements made?</td>
</tr>
<tr>
<td>Readership history</td>
<td>Who has viewed the requirements?</td>
</tr>
<tr>
<td>Motivational history</td>
<td>Why did the requirements change?</td>
</tr>
</tbody>
</table>

Table 3-1 Requirements Change Awareness Framework Overview

### 3.1 Framework Construction

This section describes the assumptions and approach inherent in the development of our framework. The framework organized our research, acting as the foundation for a coherent dialogue of requirements change awareness. While this framework is as inclusive and flexible as possible, like any conceptual tool it does not strive to be all-encompassing. We detail its propositions, assumptions, and definitions below. First, we define common terms used in the framework.

#### 3.1.1 Requirements

The IEEE guide to software requirements specification defines requirements as “a statement of some essential capability of the software to be
developed” (ANSI/IEEE 830-1984). Requirements have the properties of authorship, dates of addition, modification, deletion and completion, stated need, date of commencement and completion, and pre- or co-requirements.

3.1.2 Artefacts

An artefact is an information element that can provide information about the project and is normally the product of a stakeholder’s work. People that are part of the software development lifecycle create, modify, and remove artefacts for a project. An example of a project artefact is a document containing use cases that describe a product’s intended behaviour.

3.1.3 Software Development Roles

One can distinguish stakeholders in a project by their role in the development environment. Roles in software development projects are thereby defined as stakeholders with specific properties (Sommerville, 2007). For the purposes of this research, we designate all members of a software project as members of at least one role.

In addition, roles consist of groups of stakeholders and roles commonly follow a hierarchy within the environment. For example, a programmer would normally be subordinate to a team leader, and a team leader subordinate to a project manager. However, stakeholders have the ability to assume multiple roles; for instance, a programmer may be both client and team leader, or a client may also be a requirements engineer.

For the purposes of this framework, we identified seven common roles found in the software development lifecycle: customer, business analyst, requirements engineer, project manager, programmer, tester, and system administrator. Customers are the primary stakeholders of a project and have the ability to produce the highest levels of change to requirements. Although the customer has the ability to change technical requirements, he or she will more
commonly change business requirements. Business analysts are commonly the role that best understands how a software product will support the customer’s business model. They are responsible for understanding the benefits of the product and the cost of implementing the software. They are also responsible for managing the risk associated with development or lack of development. Requirements engineers work directly with the project requirements. For that reason, they can introduce many changes to requirements that will affect other roles in the software development project. Project managers traditionally aid in the development and fulfilment of requirements, project planning, and budgeting. Their focus is on coordinating resources throughout the implementation of a project. Programmers are responsible for writing the software. They can also be responsible for developing the architecture of the software applications that meet the requirements of a project. Testers are individuals who are responsible for validating the functionality of software implementations against the project requirements. System administrators are responsible for maintaining the customer’s information technology infrastructure. System analysts will provide requirements for the technology infrastructure and for its operation in the customer’s environment.

While many other variations of these roles may exist in various software development environments, we treat them archetypically. In the examples explored by this framework, we assume that the system administrator and the customer are part of the same client group, and the programmer, project manager, tester, and requirements engineer are part of the development group. The study also assumes that the business analyst is a third party representing the interests of the customer.

3.2 Framework Views

The requirements change awareness framework will incorporate the structure provided by Tam and Greenberg (2006). Requirements engineering
awareness consists of the ability of project stakeholders to track changes to requirements throughout the project lifecycle. Changes to requirements can be viewed from different perspectives such as Person-based, Artefact-based, and Workspace-based views. These views are consistent with the views defined by Tam and Greenberg (2006). It is possible to subdivide each view further into pieces of information that support RE awareness in that context. The rest of this section describes each of the three views and examines the information elements related to each view by identifying questions that supply answers that support increased awareness of RE information.

3.2.1 Person-Based Views

A person-based view identifies a stakeholder in a software development project. Here, we are interested in understanding the involvement of a specific person in the software development project. The components of awareness can be described along the dimensions: Who? What? When? Why? and Where? Table 3-1 enumerates the questions related to person-based information.
<table>
<thead>
<tr>
<th>Specific questions</th>
<th>Person-Based View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Who?</strong></td>
<td>Who made changes that affect a specific user?</td>
</tr>
<tr>
<td><strong>What?</strong></td>
<td>What changes to requirements did this user make?</td>
</tr>
<tr>
<td><strong>When?</strong></td>
<td>When did this user change requirements?</td>
</tr>
<tr>
<td><strong>Why?</strong></td>
<td>Why did this user change requirements?</td>
</tr>
<tr>
<td><strong>Where?</strong></td>
<td>From where did this user change requirements?</td>
</tr>
</tbody>
</table>

### Table 3-1 Awareness Data in the Person-Based View

The questions users will ask of an awareness tool regarding the properties of individuals will help us understand the types of awareness information demanded from such tools. Person-based views will help provide further detailed awareness about people working on the project. The next section includes an in-depth explanation of the questions identified above.

- **Who made changes that affect a specific user?**

Understanding who is affected by changes made to project requirements by other people is very important. This understanding helps one clarify the direction and quality of relationships between different roles, and helps one examine which roles affect and are affected by requirements change. Some roles, for instance, have the ability to modify the business requirements of a project. Business requirements are needs defined by the customer; they are normally modified only by the customer and other related stakeholders, such as business analysts and requirements engineers. Business requirements embody the customer’s needs, and their statement can generally gloss over the technical details of implementation in conception and revision. Technical requirements, however, which are the requirements related to the details of implementation,
while instrumental to achieving the project goals, must necessarily strive to meet and abide by changes to business requirements.

Consider the business requirement of a software project that specifies the need for e-mail notification on detection of a system fault. The corresponding technical requirement could originally have read to embed a mail server in the system; but, for technical reasons, later in the development cycle the requirement changed so that mail is sent from an external mail server. This change to the underlying technical requirements did not change the important feature of e-mail notification, only the roles downstream of the change. For the most part, business requirements are at the core of a project and have the ability to affect many roles. Figure 3-1 illustrates how changes to requirements can affect different roles. A number of examples (titled by initiator and type of change) can serve to demonstrate in more detail how business or technical requirements influence downstream roles.
A customer commissions a software project; during development, the customer decides to reduce the budget of the project. To reduce the budget, the customer diminishes the scope of the project by removing some of the more complex functionality of the software system. This change to business requirements causes change to technical requirements and will have a large affect on many roles in the project. In response to the change, the project manager may need to reorganize staffing resources and schedules.
**Business Analyst -- Business Change**

Consider a project in its early stages of development designed to provide inventory management for a customer. The business analyst performs a cost benefit analysis that compares developing the software in-house with the price of an off-the-shelf product. The results favour the off-the-shelf product; consequently, the business analyst recommends halting the development project and implementing the off-the-shelf solution. This change implies obvious effects on other roles in the software development environment, including the programmers who will no longer need to write any programming.

**Requirements Engineer -- Technical Change**

A requirements engineer performing a follow up interview with a customer about the layout of a graphical user interface may elicit new requirements that seem to conflict with those previously established. The requirements engineer then needs to feed this information back into the development process. This change to requirements will affect programmers and testers due to the change interface now required. In large software projects, it is common for requirements to change due to time, budget, or a new understanding of project needs.

**Project Manager – Technical Change**

A project manager responsible for the delivery of a new software product wants it to appear on the market before others. Timelines may be inflexible so that the company does not lose a competitive advantage. If the project is running behind, however, and it appears impossible to meet all requirements on time, the project manager may determine that time-to-market has a higher priority than several other features, such as multi-language support. This decision not only affects the technical requirements of software development, but also the business
requirements since the product will now have a different feature set. In addition, the change affects testers and business analysts, as language features must be removed from both the testing phase of development and the product-marketing plan.

**Programmer -- Technical Change**

A programmer tasked to implement a module that searches a database may be working within the constraints of a technical requirement that the module must query the customer database directly. However, the programmer may observe considerable delay in accessing the database. To solve this problem, the programmer performs some pre-processing of the database and creates a secondary database that runs in parallel with the primary database. Using this secondary database instead of the primary one increases the speed and accuracy of the searching by several orders of magnitude. Nonetheless, while the programmer changed the technical requirements of the project, the end-result was transparent to the customer (with the exception of the improved running time). However, this change to the requirements can influence other roles downstream of the programmer such as the system administrators. System administrators will now have to administrate two distinct databases.

**System Administrator -- Technical Change**

A customer commissions a new network monitoring system. The system administrator will need to provide the detailed requirements with regard to integration of the new software system into existing information technology systems (e-mail notification systems, logging systems, trouble ticketing systems). Changes to these systems can affect the development of new monitoring systems. Accordingly, if a system administrator changes the requirements of the project by adding or removing constraints to the deployment environment, the technical
requirements affecting roles such as programmers, who need to develop the interfaces to external systems, are also affected.

As indicated in these examples, all project requirements are related to roles; these examples explore the creation, modification, and deletion of requirements by different stakeholders and their related roles. Kontonya and Sommerville (1998) describe still other factors affecting changes to requirements not mentioned in these examples. They include errors, evolving knowledge, technical problems, changing priorities, environmental changes, and organizational changes.

For the purposes of this framework, we may summarize how changes to requirements can influence different roles: a stakeholder assuming a role can be affected by many other stakeholders’ changes. A final example illustrates this concept. Suppose Programmer A is responsible for implementing many of the requirements for a graphical user interface. Customer B who has commissioned the project decides to modify the requirements for the interface to include a new feature. This change will affect the programmer, so it is important that the programmer receive notification of the change. However, requirements engineer C must also change the requirements of the interface to support the advanced user interface. That change information also requires distribution to the programmer. Thus, we expect that when evaluating awareness tools, if a user were to inquire, “Changes made by which roles affect programmer A,” the tool would return “customer B” and “requirements engineer C” (see Figure 3-2).
Figure 3-2 Expected Results: A User Affected by the Changes of Two Distinct Roles

GUI = graphical user interface

What changes to requirements did this user make?

A user may change a project’s requirements by creating a new requirement or modifying an existing requirement. For example, requirements engineer A is documenting specifications for a new accounting system for a large bank. The requirements engineer adds Requirement A, identifying the need to support multiple languages in several different countries. The engineer also deletes Requirement B, which identifies the need for the software to run on multiple operating systems. In addition, the requirements engineer modifies Requirement C, changing the time for the accounting software to run a back-up from every 30 minutes to every 60 minutes. Our proposed framework expects that, if a user were to submit the query, “What changes to requirements have been made by Requirements Engineer A,” then that the system would return
Requirement A, Requirement B, and Requirement C. Figure 3-3 visualizes this response.

![Figure 3-3 Expected Results: Three Effects of User Changes](image)

- **When did this user change requirements?**

Understanding the motivation of people in a software development project is of the utmost importance. Generally, however, it is difficult to determine a user’s motivation for making changes to requirements without the explicit explanation of the user making modifications. Nonetheless, one can identify a user’s motivation by, for example, examining the change within a historical context. If a programmer wants to understand an update to requirements made by a requirements engineer, tracking the changes made by each user enables the programmer to request the awareness tool to return all requirements modified by
that requirements engineer and thereby discover any relationship between this and other requirements changed within a given period.

Another subtle instance of a user inquiry of the awareness tool may be to establish that a requirement was deleted directly after a meeting between the customer and a requirements engineer. The user may glean that the customer had provided the requirements engineer with new instructions to remove the requirement. The When? view, then, is critical in helping developers understand why a change occurred in their projects.

- **Why did this user change requirements?**

Information pertaining to a user’s motivations for changing requirements can be difficult to obtain through automated mechanisms. This framework proposes that user motivations be stated explicitly. The awareness tool should capture these requirement artefacts and make them available for later use. If, for example, a requirements engineer deletes a requirement for a project due to lack of budget and must now update the requirements specification document, an awareness tool should prompt the requirements engineer to record the deletion of the requirement due to budget constraints. This sub-dimension of information would provide awareness to future users interested in the reasons for the deletion of the requirement.

- **From where did this user change requirements?**

Stakeholders may be in a variety of locations. This is important to the software development lifecycle because the location of stakeholders can influence
the way they interact. Geographical proximity aids cohesion between work units in situations in which work units share a common task. This view of our framework will enumerate some of the ways that roles can be located.

Co-located roles describe stakeholders located in the same area, commonly in the same office building. Herbsleb and Moitra (2001) indicate that co-located groups have higher levels of cohesion and have better communication. When a change to a requirement occurs in a co-located environment, it is easy to communicate the change information to all relevant roles.

Roles may also have a geographic distribution. That is, a team comprised of multiple roles may have its members physically located in different office buildings. Geographical distribution can vary; the members of a team may be in the same city or country, or they may work across borders. Herbsleb and Moitra, (2001) suggest that extreme distribution can have a negative affect on the communication of a group. When requirements change in a distributed environment, requesting information from the system about the location of a team member is potentially beneficial. It can be useful in understanding cultural differences as well as in coordination across time zones.

Furthermore, the distribution of roles across physical borders may involve cultural borders. Although changes in geography and in culture are not always coordinated, different counties normally have different cultural practices, different ways of working, and different languages. When working in cross-cultural teams, it is best to encourage understanding of the different cultures, since understanding can help improve communication and trust. The extraction of the physical location of a distant user can also, therefore, suggest a contextual understanding of why a change occurs. If a requirements engineer in China makes a change to requirements by, for example, adding a larger set of fonts to a product, information contained within the Where? view may aid in deducing motivations for changes to language support requirements. In this case, the impetus could have been to provide Chinese language support for the product.
3.2.2 Artefact-Based View

Whereas the Person-based view contains the changes to and affects of requirements in relation to roles, the artefact-based view concerns understanding changes from an artefact-centric perspective. Here, we discuss the properties of software development artefacts, specifically examining the information contained within artefacts related to the requirements activities of software development projects. This data is important because changes to a development project reflect the information artefacts. One can find a logical relationship between information artefacts and the requirements they refer to and between information artefacts and software development roles. Table 3-2 summarizes the interactions involved in this view.

<table>
<thead>
<tr>
<th>Specific questions</th>
<th>Artefact-Based View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who?</td>
<td>Which user in which role changed requirements?</td>
</tr>
<tr>
<td>What?</td>
<td>Which artefacts were affected by changed requirements?</td>
</tr>
<tr>
<td>When?</td>
<td>When was the requirement changed?</td>
</tr>
<tr>
<td>Where?</td>
<td>Where in the artefact did the change occur?</td>
</tr>
<tr>
<td>Why?</td>
<td>Why did the requirement change?</td>
</tr>
</tbody>
</table>

Table 3-2 Awareness Data in the Artefact-Based View

- Which user in which role changed requirements?

When a requirement is modified, the change can affect many individuals in the workspace, so it is important to know which stakeholders initiated the change. For example, a new requirement, not previously considered or anticipated, identifies the need to interact with an additional system. This change in requirements must be transmitted to all relevant stakeholders. Examples (titled by artefact and modifying role) of the information artefacts most commonly created and modified by each archetypal role demonstrate the effects of changes.
Design Documentation – Customer

Customers create design documentation to describe their needs. On submission to one or more vendors, they will provide responses that describe how they propose to meet these needs. Changes to requirements affect design documentation; for instance, changes may occur when a project is in the early stages of development and the customer realizes the need for a specific feature in his or her software project. Design artefacts would subsequently need revision for developers to implement the new feature.

Business and Marketing Schema -- Business Analysts

The business analyst is primarily responsible for overseeing and inspecting the operations of a project, insuring that projects remain in line with organizational needs. In many cases, they work with requirements engineers to ensure that the customer’s needs are addressed in the software product design. Business analysts commonly modify artefacts such as business and marketing strategy and planning documents in the early stages of development and as they review project progress.

Requirements Specifications -- Requirements Engineer

Requirements engineers work closely with other stakeholders to add, delete, or modify requirements. Their changes usually reflect changes in requirements specification documents, which enumerate the needs of a project. A requirements engineer may change these artefacts, for instance, by gathering new requirements related to system interactions with other external systems. The requirements engineer would then update the requirements specification to reflect the changed requirements.

Project Planning Documents -- Project Manager
Project managers also work with many information artefacts. The project planning documents they work with can encompass many aspects related to project organization, such as objectives and project timelines. If a project manager changes the requirements of a project, the release date for the product, the release schedule, and other related artefacts will need modification to reflect changes.

**Source Code -- Programmer**

Programmers work with many different information artefacts, but most commonly with a product source code. Any change to a requirement that affects a programmer normally appears in the source code. For example, when the requirements for the layout of a graphical user interface have changed, the programmer will implement that change by modifying the source code of the interface.

**Summary**

The framework is crucial to identifying the rationale behind a change. Information about the user that created change provides a context to the change and, thereby, a means for finding experts related to the pertinent requirements. In most cases, this expert will be the initiator of change, since a user changing a requirement often has the most specialized knowledge about the factors involved. Thus, when queried, “Which user in which role changed project timeline information?” an awareness tool should return the name and title of the initiating role.

- Which artefacts were affected by changed requirements?

Changes to requirements have the potential to affect many aspects of the software development environment. Understanding how a requirements change will affect other elements of the project is very important, and to achieve that
understanding it is essential to record information about how the requirement changed. Some examples will help clarify this point.

Change management systems are repositories for project artefacts. For the purposes of the following discussion, we assume that such systems store their data as modification records; these artefacts reflect changes to a project. When requirements are changed, the change is recorded in the repository. For example, a customer decides to remove a feature for a product. The modules supporting that feature need deletion from the new build of the source code. Hence, the software code is affected by this change and requires update to reflect the new requirements. The programmer will make the change and modify the source code. He or she will then reinsert the modified code into the version control system repository.

Defect tracking systems also contain information artefacts. The primary function of the defect tracking system is to store changes between the program code and the requirements. Therefore, issues stored in a defect tracking system are associated to the requirements affected by the defect. Consider that in testing a new software build for clients, product testers notice that the program is not producing the correct output for a certain situation. The tester then registers a defect with the defect tracking system. The defect indicates that the software does not meet a specific requirement. Subsequently, one can find a relationship between the defect and a requirement. Defects can also relate to multiple requirements. For example, a problem or fault in a software program’s graphical user interface might affect a usability requirement, a performance requirement, and a reliability requirement.

Project documentation, such as prototypes, requests for proposals, design documents, manuals, and many other artefacts, describe the project. Any documentation related to project development can be coupled with requirements. For example, a user manual demonstrates how to use a software system. If the requirements change and some of the functionality described removed, the user
guide requires revision, thereby creating a relationship between the removed requirement and the corresponding section of the user guide that needs modification. In another situation, on elimination of a requirement from a project due to budgetary concerns, the detailed specification should change. The eliminated requirement is connected to the modified section of the specification document.

- **When was the requirement changed?**

An awareness tool should provide information regarding when requirements changed. When a requirements engineer updates requirements specifications, this view of information should reveal that the requirement about to receive modification has been more recently revised than the engineer’s notes. The temporal information thus supplies the requirements engineer with knowledge that the requirement has already received an update.

Another component of this view is that of time zones. Teams dispersed throughout the world also span time zones. In some cases, different time zones can act as a competitive advantage by providing “around the clock development” (Herbsleb and Moitra, 2001). Unfortunately, this mode of work has some disadvantages as well. It is very difficult for different stakeholders separated by time zones to communicate with each other in a synchronous manner. For the most part, stakeholders who rely on asynchronous forms of communication such as e-mail must be informed when such factors involve a requirement of interest. Querying the awareness tool, “When was the budget requirement changed?” should flag for the user any time zone difficulties, including that that the customer is out of contact about the change until the next business day.

- **Where in the artefact did the change occur?**
The sections of the artefacts modified define this view, as well as the time of modification and who modified them. When a change occurs, it requires documentation in the main requirements repository. It is important when providing awareness information to describe the modifications of an artefact, since sometimes modifications in wording a requirement can dramatically affect the meaning of the requirement.

Why did the requirement change?

This sub-dimension informs users why a requirement changed, from the perspective of the artefact, rather than from that of users. We must first understand why a change to one requirement can affect another related requirement. Suppose that Requirement A demands that, for a client database, a search return query results in less than 100 milliseconds. However, this requirement is dependent on Requirement B, which dictates a search feature in the software product. If Requirement B is removed, then little need remains for the database or its operation time (Requirement A). The two requirements are related.

It is useful to understand this relationship in comprehending why some requirements change: if Requirement B changes, then Requirement A will also change. This means that the motivation for changing Requirement A stemmed from the prior modification to Requirement B. As Madhavji (1991) detailed, our framework suggests that it is important to identify the set of items affected by change, as well as classifying and recording all related data. This identification occurs in this sub-dimension by identifying dependencies in the project, calculating affect assessments, capturing data and metadata, and making work projections.
3.2.3 Workspace-Based View

Software development is typically a group activity. Since software projects can be large and complex it is important to understand how different stakeholders interact with other stakeholders and artefacts within the project environment. Awareness information about the activities of others in the project workspace is an essential component of requirements change awareness. Here, we discuss some of the important elements of awareness with respect to the project workspace, as summarized in Table 3-3.

<table>
<thead>
<tr>
<th>Specific questions</th>
<th>Workspace-Based View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who?</td>
<td>Who has interacted in the workspace to produce requirements change?</td>
</tr>
<tr>
<td>What?</td>
<td>What interactions in the workspace produced changes to this requirement?</td>
</tr>
<tr>
<td>Where?</td>
<td>Where in the workspace did the requirements change occur?</td>
</tr>
<tr>
<td>How?</td>
<td>How does this user interact in the workspace to make requirements change?</td>
</tr>
</tbody>
</table>

Table 3-3 Awareness Data in the Workspace-Based View

- **Who has interacted in the workspace to produce change?**

Figure 3-1 illustrates the roles in the person-based view likely to affect other users by modifying requirements. Adding finer granularity to the concept introduced by that diagram, this view explores the interacting roles that produce changes to requirements. Figure 3-4 begins an analysis of this view by exploring the archetypal interactions in the workspace.
Changes to requirements can stem from any number of motivations, but it is likely that changes will occur more frequently when roles interact during collaboration. The context of the decision to change a project specification is important to awareness: a change resulting from the interaction of the customer and project manager could be more far reaching than a change discussed by two programmers. Hence, it is crucial to a promotion of awareness to capture these change initiating interactions at the time of the change. To restrict all project-based communication to a development bulletin board, thereby retaining the background to any major (or minor) modifications to project requirements for later review serves as an example.

- What interactions in the workspace produced change to this requirement?
This view relays information about the types of activities in which users engage to produce changes to requirements. Group activities include the major types that McGrath and Hollingshead (1994) describe: generating ideas, planning tasks, solving problems, and making decisions.

**Generating Ideas**

A group of programmers working together on the high level architecture of a software development project use a whiteboard to express their ideas and to define the prominent interfaces between software modules. In expressing these concepts, they realize the need for several modules to communicate to an external system. This group activity produced new ideas that led to a change to the requirements of the project.

Generating ideas may also affect requirements when customers and requirements engineers participate in brainstorming sessions in focus groups. During focus groups, requirements are often elicited because of group discussions.

**Planning Tasks**

A project manager and customer work together to plan the scheduling of a product. The project has several milestones, so the requirements for the programmers include completing their tasks before these milestones. The activity of planning the schedule for the project thereby affects the requirements for the programmers.

In a different example, business analysts and customers can work together to design a product to meet business needs. This can include planning the implementation of certain features that offer a competitive market advantage, or removing features that are no longer desirable. This planning has a direct affect on roles downstream of potential changes to requirements: when a feature is dropped, programmers no longer need to invest time in implementing it.
Solving Problems

A group of programmers working with requirements engineers on implementing customer requirements define and establish new technical requirements that will affect other developers in the project. The different roles collaborate on solving problems related to requirements. The results of these interactions introduce changes to the requirements specification.

Making Decisions

Customers work with business analysts to make decisions on the priority of requirements. If a customer decides that one requirement is of higher priority than another, the project timeline and staffing resource requirements are affected, favouring the development of the higher priority requirements.

Requirements engineers also make many decisions that affect project requirements. For example, suppose a requirements engineer is developing a survey to elicit requirements from the customer. The requirements engineer must make critical decisions on which questions to include in the survey and on the method for interpreting the results. These factors will affect the development of requirements for the project.

How does this user interact in the workspace to make change?

How a person makes changes in the workspace is also important. Interactions may be either synchronous or asynchronous exchanges with other users in the workspace and affect change in the workspace. For example, a requirements engineer performing an interview to elicit requirements is engaged in a synchronous exchange that will affect project requirements. Alternatively, testers using a bug manager to track problems with source code defects can communicate issues to programmers and other developers by posting a bug report.
This information asynchronously affects requirements by enumerating newly found issues that will later require reengineering of the software.

- Where in the workspace did the change to requirements occur?

This question is defined by which artefacts were modified, by when they were modified, and by whom they were modified. Changes require documentation in the main requirements repository. However, for changes made to other artefacts, such as project planning documents, it is important to understand where the change occurs if one is to understand how a change to requirements affects the rest of the workspace.

3.3 Information Elements

An important goal of this framework is to apply it to the evaluation of RE tools and gauge their level of support for providing awareness information. To do this, we relate concepts of awareness to pieces of information. These pieces of information, so-called information elements (Tam and Greenberg, 2006), characterize which activities should be monitored within a project to capture change information about requirements. Tam and Greenberg describe information elements as “specific pieces of information that a person would require in order to keep up with events as they occur in a collaborative real-time workspace” (Tam and Greenberg, 2006).

Since our framework rests on the work of Tam and Greenberg (2006), we use their evaluation methodology as a guideline for evaluating our set of tools. They suggest mapping informational elements to the questions in the framework. The information elements provide answers to the questions posed in the requirements change awareness views.
Two information elements will represent the Where? dimension of our framework: *geographical location history* and *workspace location history*. We will evaluate the tools assessing the level of support they give for these elements. The geographical location history expresses how well a tool provides a concept of locality of stakeholders. Ideally, the tools take into account that stakeholders may be located in different physical environments. The tool should also supply information on the physical distribution of stakeholders and the environment from which each user operates. For example, if a developer lives in India and a project manager in Canada, then the tool should report the pertinent differences, including different time zones and different organizational centres. Workspace history is different from physical locality. It refers to a location in the project work product. That is, when a programmer makes a change to the source code of a project, the workspace location will reference the exact area of the code base repository to which the programmer made the change.

The Who? dimension of our framework will consist of *change affect history*, *interaction history*, *authorship history*, and *readership history*. Change affect history refers to the ability to show which stakeholders the addition, removal, or modification of requirements will affect. This provides traceability information between requirements, stakeholders, and their work products. Interaction history refers to the ability to understand who has collaborated on changing requirements. For example, if a requirement were modified due to a meeting between a business analyst and the customer, the interaction history should indicate that the changes to the requirements were the result of their interaction. This type of information is important in understanding the motivations behind changes to requirements, as well as in providing insights to ownership of requirements. Authorship history indicates who specifically made a change to a requirement. Readership history is information documenting who has viewed the requirement. This is an important element of awareness, since it aids in gauging a stakeholders’ level of shared understanding. If a project manager
wants to know if the programmers have read the updated requirements specification they could check the readership history of the requirement. This grants the project manager information about what knowledge the programmers have.

The What? dimension of our framework consists of the changes made to the requirements. The main element of this dimension is the change history in the project. By examining the history of the changes to requirements, it is possible to understand the work of specific users. It is important to track what changes the user makes to understand how that will affect the rest of the project.

The How? dimension of our framework deals with outcome history and process history. The realm of the outcome history is understanding how the requirement changed. The outcome history identifies the current status of the requirements, such as if the requirement has begun, is in progress, or is complete. These elements are essential for providing stakeholders with project awareness information and preventing duplication of efforts. The process history refers to how the requirement was changed. It indicates how, incrementally, the requirement evolved from its previous state. Understanding how the requirement changed refers to the ways in which stakeholders update requirements.

The When? dimension of our framework refers to the temporal characteristics of requirements change. The tools should provide an event history that indicates the exact time of modifications to the requirement. Providing a chronology of changes to the requirements is important to increasing awareness. For example, a stakeholder might want to know when a requirement was created and who made the last change to it.

The Why? dimension of our framework concerns understanding the rationale of requirements change. Motivational history refers to the individual reasoning for the change to the requirement (Tam and Greenberg, 2006). This information provides stakeholders with awareness regarding the reasons for modification of the requirement. For example, when a developer makes a change
to a project requirement, they receive a prompt to enter a comment as to why they made the change. Table 3-5 indicates the relationship between the framework dimensions and the information elements.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Information Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who?</td>
<td>Change affect history</td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
</tr>
<tr>
<td>What?</td>
<td>Change history</td>
</tr>
<tr>
<td>When?</td>
<td>Event history</td>
</tr>
<tr>
<td>Where?</td>
<td>Geographical location history</td>
</tr>
<tr>
<td></td>
<td>Workspace location history</td>
</tr>
<tr>
<td>How?</td>
<td>Outcome history</td>
</tr>
<tr>
<td></td>
<td>Process history</td>
</tr>
<tr>
<td>Why?</td>
<td>Motivational history</td>
</tr>
</tbody>
</table>

Table 3-2 Framework Views and Information Elements

3.4 Summary

This chapter has described different views of awareness combined into a framework intended to capture significant aspects of requirements change awareness. The framework divides requirements change awareness into person-based views, artefact-based views, and workspace-based views. The framework examines elements such as the nature of changing requirements and their relationship to project roles. The framework also provides a description of
information sources related to requirements changes as well as of group interactions that can spawn change. Table 3-6 grants an overview of the three different views and the related information elements.

<table>
<thead>
<tr>
<th>Specific Questions</th>
<th>Workspace-based View</th>
<th>Person-based View</th>
<th>Artefact-based View</th>
<th>Information Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Who?</strong></td>
<td>Who has interacted in the workspace to produce change?</td>
<td>Changes made by which roles affect this user?</td>
<td>Which user in which role changed requirements?</td>
<td>Change affect history</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interaction history</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Authorship history</td>
</tr>
<tr>
<td><strong>What?</strong></td>
<td>What interactions in the workspace produced changes to this requirement?</td>
<td>What changes to requirements did this user make?</td>
<td>Which artefacts were affected by changed requirements?</td>
<td>Outcome history</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Readership history</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change history</td>
</tr>
<tr>
<td><strong>When?</strong></td>
<td></td>
<td>When did this user change requirements?</td>
<td>When did the requirement change?</td>
<td>Event history</td>
</tr>
<tr>
<td><strong>Where?</strong></td>
<td>Where in the workspace did the change occur?</td>
<td>From where did this user change requirements?</td>
<td></td>
<td>Geographical location history</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Workspace location history</td>
</tr>
<tr>
<td><strong>How?</strong></td>
<td>How does this user interact in the workspace to make changes?</td>
<td></td>
<td></td>
<td>Process history</td>
</tr>
<tr>
<td><strong>Why?</strong></td>
<td></td>
<td>Why did this user change requirements?</td>
<td></td>
<td>Motivational History</td>
</tr>
</tbody>
</table>
Tam and Greenberg’s (2006) framework shares many components with our framework, including the provision of different perspectives on changes from artefact-, people-, and workspace-based views. We list many other common information elements in Table 3-6. Unfortunately, Tam and Greenberg’s framework was not sufficient to describe requirements change awareness. In this summary, we review the differences between our framework and that of Tam and Greenberg.

The first element that we removed from their framework when constructing ours was gaze history. Tam and Greenberg (2006) used the concept of gaze history to understand what people looked at in the workspace. This concept passed into their framework from an earlier framework developed by Gutwin and Greenberg (2001) for assessing awareness in real-time distributed groupware. The ability to know what a person is looking at in a real-time environment can provide powerful awareness clues. However, in an asynchronous requirements management situation, with less emphasis placed on real-time interaction, the benefits diminish. In any case, gaze history and readership history offer similar information when taken in an asynchronous context.

Another component extracted from Gutwin and Greenberg’s (2001) framework on awareness in real-time distributed groupware was that of presence history. Presence history describes who has looked at an artefact, with whom people interact, and who has been in the workspace. We thought that readership history would offer enough information about who has viewed an artefact. Information about who has been in the workspace is more suited to a real-time groupware situation than a requirements management tool. We did think, however, that interaction history was important in understanding the context for changes. Accordingly, we created an information element called interaction history that shows how people interact in the creation, modification, or removal of requirements.
Another difference in the two frameworks lies in the interpretation of location history. Tam and Greenberg (2006) use location in a very specific sense to identify where in the workspace a person is working. However, we thought it necessary to provide a richer definition of locality and created two distinct information elements. We capture different perspectives on locality by using workspace location history and geographical location history. Adding this distinction makes it possible to offer richer awareness information specifically for distributed teams.

Tam and Greenberg (2006) use edit history to describe to which parts of the workspace people have made changes. This makes sense from a groupware perspective, but is less useful in a requirements management setting where we focus on understanding requirements changes. Instead, we created a more focused information element called requirements change history; this element provides the history of changes to requirements, making it possible to understand the history of modifications and to propose an evolutionary context for requirements.

The last element of Tam and Greenberg’s (2006) framework that we thought was unnecessary was the concept of identity. Identity provides information about who has changed artefacts. Understanding who has made changes is extremely important to both frameworks, but we thought it well captured in authorship history. In fact, the way that Tam and Greenberg use identity history is an instance of authorship history.

The final modification that we made to Tam and Greenberg’s (2006) framework was that of change affect history. Change affect history captures information related to requirements traceability and dependencies. Since requirements have such potential to affect other downstream elements of a software project, a separate information element dedicated to describing the properties on change affect analysis was essential.
Chapter 4

Application of the Framework to the Evaluation of Three Requirements Engineering Management Tools

Here, we discuss the application of our framework to the evaluation of three state-of-the-art requirements management tools. One purpose of the framework is to permit researchers to compare tools that provide awareness support for requirements. By evaluating the tools based on the framework, practitioners will find it easier to identify those tools that best support requirements change awareness.

4.1.1 Tools Evaluated

The evaluation will examine three different requirements tools: RequisitePro, DOORS, and CaliberRM. These three RE applications are the most widely used enterprise tools in current industrial practice.

4.1.2 Evaluation Procedure

The author evaluated the tools using the Software Engineering Global interAction Laboratory (SEGAL) computing facilities, located at the University of Victoria. The applications were deployed and tested within two months between 25 November 2007 and 4 January 2008. All of the tools required setting up a client and a server with servers deployed on Windows Server and clients running
on Windows XP. After installation, templates were loaded into the systems. The templates were similar in complexity and structure. The next step was to create sample users such as project managers and developers. After the project environment was set up, the evaluation required performing six different case tasks that changed the state of the project requirements. The tasks involved creating a new requirement in each project and defining three traceability links to other project requirements. It was also necessary to set the attributes of the new requirement, such as the status and the priority. The actual requirement description was not significant since the goal of the exercise was to examine how the tools reacted to changes to requirements independent of the needs that the requirement described. After each task, we evaluated the tool’s level of awareness support corresponding to the defined information elements. Table 4-1 serves as a checklist for evaluating the tools.
<table>
<thead>
<tr>
<th>Category of Questions</th>
<th>Information Elements</th>
<th>Caliber RM</th>
<th>RequisitePro</th>
<th>Doors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where?</td>
<td>Geographical location history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Workspace location history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Who?</td>
<td>Change affect history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>What?</td>
<td>Change history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How?</td>
<td>Outcome history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>When?</td>
<td>Event history</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Why?</td>
<td>Motivational history</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1 Check List for Tool Evaluation

### 4.2 Evaluation Use Cases

We designed a number of use cases to capture situations that require awareness and to enable us to verify aspects covered by the framework. The use cases exemplify typical types of activities done while managing requirements.
such as modifying requirements and examining requirements (Sommerville and Sawyer, 1997) and exemplify common interactions that users have with requirements management tools. Table 4-2 indicates the correspondence between use cases and information elements. We rated the results of each use case with a three point scoring scale that indicated the level of support each tool provided for the information element of our framework.

<table>
<thead>
<tr>
<th>Real world RE use cases</th>
<th>Use cases in evaluation</th>
<th>Information Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modifying Requirements</strong></td>
<td>Adding requirements</td>
<td>Geographical location history</td>
</tr>
<tr>
<td></td>
<td>Changing requirements</td>
<td>Change history</td>
</tr>
<tr>
<td></td>
<td>Removing Requirements</td>
<td>Workspace location history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Change affect history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interaction history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authorship history</td>
</tr>
<tr>
<td><strong>Examining Requirements</strong></td>
<td>Checking status</td>
<td>Outcome history</td>
</tr>
<tr>
<td></td>
<td>Checking coverage</td>
<td>Process history</td>
</tr>
<tr>
<td></td>
<td>Seeking information related to requirements change</td>
<td>Event history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Readership history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Motivational history</td>
</tr>
</tbody>
</table>

Table 4-1 Use Cases Corresponding to Information Elements

We structured the use cases to demonstrate if the tools provided support for each requirements change awareness information element according to the
framework. The use cases had a primary actor responsible for interacting with the RE tool. A pre-condition was defined for each use case stating background information about user interactions with the RE tool. The use case also had an explicit goal that the actor needed to achieve, such as adding a new requirement to the requirement database. During the evaluation, a checklist indicated whether the use case demonstrated support for a requirements change awareness information element. In our summaries, we provide detailed information about use cases and an example checklist. Each check was ranked with either a “√,” which indicated that the tool explicitly supported the information element, a “~,” which indicated that the tool only partially supported the information element, or an “x,” which meant that the tool offered no support for the information element.

**Use Case 1: Adding Requirements**

*Primary Actor: Project manager (or business analyst or developer)*

**Needs 1.1**

Pre-condition: The identification of a new requirement

Goal: The project manager needs to update the project documentation with a new requirement, so he or she creates a new requirement using the requirements management tool.

**Use Case 1: Checklist**

**Check 1.1**

*Did the tool record the geographic location of the project manager when the change was made?*

√ = The tool provided the geographical location of the project manager that added the new requirement.

• Example: Location of the user is 1600 Amphitheatre Rd, San Francisco, CA, USA

~ = The tool provided general geographical location information, but not the exact location on the project manager.
• Example: Location North America
• Example: The administrator can see the location information, but not the other users in the application.

\[ x = \text{The tool furnished NO geographical location information about any of the project members.} \]

• Example: No location data available for any users

**Check 1.2**

*Did the tool indicate where in the workspace (which elements of which documents) the requirements changes were made?*

\[ \checkmark = \text{The tool visually indicated where in the project documents new requirements appear.} \]

• Example: A map of the project workspace showing the complete document (e.g., SRS) modified by adding a requirement; also changes are highlighted in each document.

\[ \sim = \text{The tool provided some indication that a new requirement was added to project documents, but does not visually show the exact location of the new addition.} \]

• Example: The tool shows that changes occurred to the requirement specification, but not exactly where the modification took place.

\[ x = \text{The tool provided NO information about the location of the addition of requirements in project documents.} \]

• Example: No notifications occur when requirements are added.

**Check 1.3**

*Did the tool identify all stakeholder and respective roles and all project artefacts affected by the change?*

\[ \checkmark = \text{The tool provided a list of all the stakeholder names, roles, and artefacts affected by adding a new requirement to the project.} \]

• Example: Changes to requirement A will affect:
  
  o Andrew Swerdlow (Project Manager),
• requirement B
  • Source Code Module
• requirement C
  o James Chisan (programmer),
• requirement D
  • Source Code Module delta
  o Nav Jagpal ( tester)
• requirement E

~ = The tool provided information about one of the following: stakeholder names, roles, and artefacts affected by adding a new requirement to a project. The tool did not yield all three sources of information.
  • Example: Changes to requirement A will affect:
    o Andrew Swerdlow, James Chisan, Nav Jagpal
  • Example: Changes to requirement A will affect:
    o requirement A, B, C, D, E

x = The tool did not provide a list of all the stakeholders names, roles, and artefacts affected by adding a requirement to a project.
  • The tool did not supply any information about those people or requirements affected by changes.

Check 1.4

Did the tool record the names, roles, and communication artefacts of those users involved in interactions that caused the change?
√ = The tool provided a list of names, roles, and artefacts related to communication such as chat logs, or bulletin board postings.
  • Example: On proposal of a change, an online discussion took place in the workspace between Andrew Swerdlow (Project Manager) and James Chisan (programmer) about the nature of the change. That discussion and information about the people involved is linked to the requirement description.
~ = The tool provided some of the names and roles of the people involved in adding the requirements. However, the tool might not list all the stakeholder names and roles and did not supply any contextual information about their interactions in the workspace.

- Example: On proposal of change in a requirement, an online discussion took place in the workspace between Andrew Swerdlow and James Chisan about the nature of the change. That discussion and information about the people involved appear in the requirement history; however, no role information was saved about the conversation.

x = The tool provided NO information about the stakeholder names and roles or their interactions that spawned changes to the requirements.

- No information is available information regarding how people made the changes to the requirements.

Check 1.5

Did the tool record the name and role of the user that changed the requirement?

√ = The tool provided the name and the role of the user that added the requirement.

- Example: Andrew Swerdlow (Project Manager) added requirement A.

~ = The tool supplied the name or the role of the user that added the requirement, but not both.

- Example: Andrew Swerdlow added requirement A.

x = The tool offered NO information about the user that added the requirement to the project.

- Example: No information is available on who added requirement A.
Check 1.6

*Did the tool record information about all the changes a user has made?*

\( \sqrt{\text{ }} = \text{The tool provided a complete list of all the changes each user made.} \)

- Example: Andrew Swerdlow made 26 changes to the project; see the list of changes (C1, C2, C3, \ldots, C26).

\( \sim = \text{The tool furnished some information about the changes a user made, but not a complete list.} \)

- Example: Andrew Swerdlow has made changes to requirements A, B, C (does not give all changes, only requirements he has changed).

\( x = \text{The tool offered NO information about what changes users made.} \)

- Example: No information is available regarding what requirements a user changed.

Use case 2 and 3 also involve modifying requirements, but examine modifying existing requirements and removing requirements from a project. In these use cases, the checks will be the same as in use case 1.

**Use Case 2: Removing Requirements**

*Primary Actor: Project manager (or business analyst or developer)*

**Needs 2.1**

*Pre-condition: A requirement requires removal from the project.*

*Goal: The project manager is to identify and delete an obsolete requirement.*

**Use Case 3: Changing the Requirements**

**Needs 3.1**

*Pre-condition: A requirement needs modification.*
Goal: The project manager needs to change the project documentation by modifying a requirement, so he or she identifies the requirement that needs modification and changes it using the requirements management tool.

Use Case 4: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 4.1

Pre-condition: The project has changed status.

Goal: The business analyst needs to know the status of the project, so he or she uses the tool to examine the current state of requirements in the project and ensure that requirements have been satisfied.

Use Case 4 Check List

Check 4.1

Did the tool furnish the current status of the requirements and can the business analyst check each of the requirements in the project and confirm that they are covered by components of the implementation?

√ = The tool provided the status of each requirement and a project status overview linked to implementation artefacts. All requirements should have a defined status.

• Example:
  o Requirement 1 = In Progress and covered by UML (Unified Modeling Language) Diagram 1.
  o Requirement 2 = Complete and covered by CVS (Concurrent Versioning System) repository Req2.
  o Requirement 3 = On hold and is covered by Use Case 2.
  o Requirement n = Urgent and is covered by external interface specification.

~ = The tool allowed users to obtain the status of a requirement, but did not require the status to be set. The tool did not have a project status
overview function or might not link the implementation artefacts to the requirements.

- Example: (each requirement must be manually examined, and status is not mandatory)
  - Lookup Requirement 1
    - Requirement 1 = N/A
  - Lookup Requirement 2
    - Requirement 2 = Complete and is covered by CVS repository Req2

x = The tool yielded NO information about project requirements status.

- Example: No status information on requirements.

Use Case 5: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 5.1

Pre-condition: Requirements have been modified and the software developers are alerted of the changes.

Goal: The software developers want to review the requirements because they have information regarding recent changes to the requirements; they use the requirements tool to verify changes to the requirements.

Use Case 5: Check List

Check 5.1

Did the tool record the exact time and date of modification to the requirement?

√ = The tool provided the exact time and date of the requirement modification.

- Example: Requirement A received modification at 10:37pm PST on 19 September 2007.

~ = The tool offered the date, but not the exact time.
• Example: Requirement A was modified on 19 September 2007.
  x = The tool gave NO information regarding modifications.
• Example: No information is available regarding when the requirement was modified.

Use Case 6: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 6.1
Pre-condition: Several different stakeholders have modified the project requirements.
Goal: The software developer is trying to find out who was working on implementing the requirement before the change, so he or she uses the requirements tool to find who modified the requirement.

Use Case 6: Check List

Check 6.1
Did the tool record how the requirement had changed; that is, does the tool show the evolution of the requirement and the different revisions to the requirement?
√ = The tool provided an incremental view of how the requirement evolved from its previous state.
~ = The tool offered some different versioning information, but at a low granularity.
• Example: Show last revision on Requirement A (only the current version and the last version Rec-A-Version-2.0, Rec-A-Version-2.1).
  x = The tool yielded NO version tracking of requirements.
• The tool provides no version information.

**Check 6.2**  
*Did the tool provide names and roles of other stakeholders who viewed the requirement?*

√ = The tool gave the name and the role of every user that viewed any project requirement.

• Example: Show all users who viewed Requirement A = Andrew Swerdlow (Project Manager), James Chisan (programmer), Dana Damian (Manager).

~ = The tool provided the name or the role (but not both) of users that viewed the requirement.

• Example: Show all users that viewed Requirement A =  Andrew Swerdlow, James Chisan, Dana Damian

x = The tool granted NO information about users that viewed the requirements.

• The tool did not list any readers of the requirement.

**Check 6.3**  
*Did the tool record an explanation of changes?*

√ = The tool enforced mandatory explicit comments from users making the changes.

• Example: A requirement that has been modified three times has three comments:
  
  o Nav Jagpal //I have added this requirement at the customer’s request.
  
  o Andrew Swerdlow //I am changing this requirement to fit our new time line, since the project was delayed due to the price of oil.
James Chisan //I have just stated to work on the requirement and have changed the status to in progress.

- The tool provided optional comments on changes.
  - Example: A requirement that has been modified three times has one comment:
    - James Chisan //does anyone know why this was added????
- The tool furnished no formal comment on changes by users making the changes.
  - Example: No comments

Once the checks had occurred, the results were recorded in a table. Table 4-2 describes the relationship between the information elements, the use cases, and the checks.
<table>
<thead>
<tr>
<th>Category of Questions</th>
<th>Information Elements</th>
<th>Use Case Check Coverage</th>
<th>Use Case Needs Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Where?</strong></td>
<td>Geographical location</td>
<td>Check 1.1, 2.1, 3.1</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td></td>
<td>history</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Workspace location</td>
<td>Check 1.2, 2.2, 3.2</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td></td>
<td>history</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Who?</strong></td>
<td>Change affect history</td>
<td>Check 1.3, 2.3, 3.3</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
<td>Check 1.4, 2.4, 3.4</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
<td>Check 1.5, 2.5, 3.5</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
<td>Check 6.2</td>
<td>Use case 6</td>
</tr>
<tr>
<td><strong>What?</strong></td>
<td>Change history</td>
<td>Check 1.6, 2.6, 3.6</td>
<td>Use cases 1-3</td>
</tr>
<tr>
<td><strong>How?</strong></td>
<td>Outcome history</td>
<td>Check 4.1</td>
<td>Use case 4</td>
</tr>
<tr>
<td></td>
<td>Process history</td>
<td>Check 6.1</td>
<td>Use case 6</td>
</tr>
<tr>
<td><strong>When?</strong></td>
<td>Event history</td>
<td>Check 5.1</td>
<td>Use case 5</td>
</tr>
<tr>
<td><strong>Why?</strong></td>
<td>Motivational history</td>
<td>Check 6.3</td>
<td>Use case 6</td>
</tr>
</tbody>
</table>

Table 4-2 Framework Views and Use Case Coverage
4.2.1 Problems Encountered during Evaluation

We encountered several problems while doing the evaluations. One of the challenges with applying the framework was quantifying the level of support each tool provided for each information element. In many cases, the tools granted some support for the information element, but it was difficult to obtain a high level of granularity in representing their exact levels of support.

We also encountered challenges in simulating a real project environment without having multiple users in different roles and locations. This was problematic since it was difficult to gauge the effectiveness of the awareness mechanisms.

We had other issues related to complex licensing and dedicated server back end infrastructure. These made it difficult to deploy the tools and required working with the software manufacturers and the University of Victoria information technology department. After resolution of these difficulties, we encountered several other issues related to the usability of the tools. Most of the tools had a very steep learning curve and required a training period. Becoming familiar with the tools required reading the documentation, trying examples, and working with the vendor support staff. These variables could affect gauging the effectiveness of the awareness support that each tool yielded.

We also used standard templates for each tool; the templates were similar with respect to requirements hierarchy, types of requirements, number of modules, priorities, attributes, and users. We found some variance in the number of requirements in each project and the amount of traceability links. See Table 4-1 for a comparison of the project templates.
4.3 Tool Evaluation: *CaliberRM*

The first tool evaluated was *CaliberRM* 2006, a software application available from Borland Corporation. The application is designed to support requirements management. It stores all project requirements in a centralized repository. The software also provides traceability linking between requirements and project artefacts. *CaliberRM* allows users to analyze the affect of changing requirements by visualizing the traceability relationships. *CaliberRM* also allows access to the repository through a web interface, making the requirements database highly accessible.

*CaliberRM* furnishes a flexible way to define different types of requirements, including business requirements and testing requirements. Users can define requirement types to allow for customized grouping of requirements, offering great flexibility and allowing customization to unique environments.
*CaliberRM* also incorporates the concept of requirements attributes. An attribute is a characteristic that defines a requirement. An example of a requirement attribute is the priority status of a specific requirement. The system can implicitly link the attributes to the requirement or users may explicitly so do.

*CaliberRM* allows several visualizations of requirements, such as in a requirements graph (Figure 4-13) or in traceability matrix form (Figure 4-14). The requirements graph is a model of all the project requirements in a graphical form with nodes and edges showing relationships between requirements. This view is useful for providing an overview of the project requirements and their specific characteristics. The traceability matrix view allows users to see relationships between requirements and other artefacts. This view is more important for understanding what affect changes to requirements might have on other requirements.

The *CaliberRM* software also provides some collaboration tools; for example, the “Group Discussion Feature” allows users to add and modify comments related to each requirement. This discussion tool is available either at a global level for the entire project or for each requirement. The discussion feature furnishes a means for stakeholders to collaborate asynchronously on defining requirements.

**Pre-Use-Case Testing Notes**

We tested Borland's *CaliberRM* 2006, running on Microsoft Windows Server 2003, on 25 November 2007. For the purposes of this evaluation, testers used the templates packaged with the software. The first step involved setting up environment information regarding users and departments. We created three different departments, Business Development, Software Engineering, and Project Management, and associated projects with the groups. Figure 4-2 shows how to create new users. The next step was to create roles for the new user see Figure 4-3.
Figure 4-2 CaliberRM: Creating Users
Figure 4-3 CaliberRM: Creating Users Role

The software required contact information and geographical location information for each user. We then added the users to their respective groups. Figure 4-4 shows how to add user specific information.
Use Case 1: Adding Requirements

Primary Actor: Project manager (or business analyst, or developer)

Needs 1.1

Pre-condition: A new requirement has been identified.

Goal: The project manager needs to update the project documentation with a new requirement, so he or she creates a new requirement using the requirements management tool.

The project manager logged into the client see Figure 4-5.
In this example case, we defined three different types of requirements types: business requirements, user requirements, and functional requirements. Figure 4-6 depicts the project manager adding a new business requirement.
The next step was to select the requirement class and create a new child requirement. We provided a name and a description for the requirement, in this case:

ATM’s shall print a receipt after each transaction. Otherwise, a message appears on the ATM screen indicating that a paper receipt is not able to be printed. Each of the above transactions will be documented on paper and a printed copy of the paper receipt will be dispensed to the customer at the conclusion of each transaction.

It was also necessary to provide such information as the owner (by default, the creator), the status, and the priority see Figure 4-7.
Next, we established traceability links between requirements. In this case, the project manager assigned the requirement a traceability link from one requirement and two links to another. Figure 4-8 depicts adding traceability links.

---

**Figure 4-7 CaliberRM: Creating a Requirement**

ATM's shall print a receipt after each transaction. Otherwise, a message appears on the ATM screen indicating that a paper receipt is not able to be printed. Each of the above transactions will be documented on paper and a printed copy of the paper receipt will be dispensed to the customer at the conclusion of each transaction.
Figure 4-8 CaliberRM: Assigning Traceability Links

The next step was for the project manager to provide information regarding the requirements features. The user also had the option to supply a due date for the requirement. Figure 4-9 shows adding details to the requirements.
Next, the project manager decided whether to add approvals for compliance, funding, and management, here selecting all of the approvals. Figure 4-10 shows adding approvals chains to the requirement.
Once we had created the basic requirement entity, we could assign the requirement to a user, in this case the project manager. Figure 4-11 depicts assigning a requirement to a specific department or users.
Figure 4-11 CaliberRM: Assigning a Requirement

The user then can add artefacts to a requirement. Here the project manager added a photograph of an ATM and notes on the ATM, as shown in Figure 4-12. To finish, the user must save changes and add comments.
Check Use Case 1 Results

Check 1.1

Did the tool record the geographic location of the project manager when the change was made?

The tool associates geographical locations with the users. By default, however, the tool did not display this information in the change history or in its reporting features. The same occurred with use case 3. In use case 2, when a requirement was deleted a record was not stored in the project’s reports or history. No geographical information was retained, therefore, for the user who deleted the requirement. Thus, the tool provided some awareness information for this element.
Check 1.2

Did the tool indicate where in the workspace (which elements of which documents) the changes were made?

The tool did not provide explicit support for mapping changes to elements in the workspace. The same is true for use case 3. In use case 2, the tool did not record the location information about deleted requirements in the project’s reports or history. The tool does not furnish awareness information for this element.

Check 1.3

Did the tool identify all stakeholders and respective roles affected by changes to the project and did the tool create a list of all project artefacts affected by the change?

CaliberRM provided an assessment of how the new requirement related to other requirements using several visualizations such as traceability diagrams and matrices. However, the tool did not enumerate all the stakeholders associated with those requirements linked to the new requirement. The same is also true for use case 3. In use case 2, the tool made no record of the deleted requirement stored in the project’s reports or history; however, it did remove all the traceability links to and from the deleted object. Thus, the tool granted some awareness information for this element. See Figure 4-13 and 4-14.
Figure 4-13 CaliberRM: A Traceability Diagram Plots Linked Requirements
Check 1.4

Did the tool record the names and roles of those users involved in interactions that caused the change?

The tool kept a history of all the users that added or modified requirements. However, these records indicated only the name of the user that added or changed the requirement in the application. They did not associate multiple users to a single change record. For use case 2, the tool stored no information for each requirement or related interaction information. In use case 3,
the tool did not provide all of the users involved in making the changes to the requirements. It did show all the users that modified the requirement using the *CaliberRM* tool and it did have a bulletin board feature for discussing requirements; the latter offered some limited information on discussions related to the requirement. See an example in Figure 4-15. Thus, the tool granted some awareness information for this element.

Figure 4-15 CaliberRM: A Change Record

Check 1.5
Did the tool record the name and role of the user who changed the requirement?

The tool indicated the role of the user who added the requirement. This is found when the user points the mouse at the entry in the change record. For use case 2, the tool did not store any information about the stakeholder roles involved in the change. However, in use case 3 the tool provided the name and role of the user who modified the requirement. See Figure 4-16. Thus, the tool offered awareness information for this element.

![Figure 4-16 CaliberRM: Change information](image)

Check 1.6

Did the tool record information about all the changes a user made?

For use cases 1 and 3, the tool showed the changes that the user made, but not a list of all the changes. In use case 2, on deletion of a requirement no record was stored in the project reports and history. Thus, the tool provided some awareness information for this element.

Use Case 2: Removing Requirements

Primary Actor: Project manager (or business analyst or developer)

Needs 2.1

Pre-condition: A requirement needs to be removed from the project.

Goal: The project manager identifies and deletes an obsolete requirement.
After logging into the application and identifying the requirement to remove, the project manager chooses to delete the “Receipt Req.” Figure 4-17 depicts the processes of removing the requirement.

Figure 4.17 CaliberRM: Identifying and Deleting Requirements
Use case 4: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 4.1

Pre-condition: The project has changed status.

Goal: The business analyst needs to know the status of the project, so he or she uses the tool to examine the current state of requirements in the project and to ensure satisfaction of requirements. The business analyst will log into the tool, as depicted in Figure 4-18. The business analyst can then check the status of the requirements by selecting the requirement for the project hierarchy and viewing the details of the requirement, as depicted in Figure 4-19. The business analyst also has the option to access the project reports that give an overview of all requirements and their status in the project.
Figure 4-18 CaliberRM: Checking the Status of Requirements in the Hierarchy
Figure 4-19 CaliberRM: Checking the Status of Requirements from the Report

Check Use Case 4 Results

Check 4.1

Did the tool provide outcome histories and can the business analyst check each of the requirements in the project and confirm their coverage by components of the implementation?

The tool did provide detailed outcome histories in the form of status flags and the business analyst could map each requirement to different artefacts and users. Thus, the tool yielded awareness information for this element.

Use case 5: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 5.1

Pre-condition: Requirements have been modified and the software developers are alerted of the changes.
Goal: The software developers want to review the requirements because they have information regarding recent changes to the requirements. They use the requirements tool to see how the requirement has been modified. The developers will log into the application, as shown in Figure 4-20. The developer can then review the history of the requirement by selecting the requirements history property, as shown in Figure 4-21. The developer can also select the requirement and execute a comparison between the current state of the requirement and its previous state.

Figure 4-20 CaliberRM: Reviewing the Requirements History from the Hierarchy
Figure 4-21 CaliberRM: Reviewing the Requirements History Report

Check Use Case 5 Results

Check 5.1

Did the tool record the history of how the requirement changed; that is, does the tool show the evolution of the requirement and all of the different revisions to the requirement? Does the tool also record the name and role of the stakeholder that revised the requirement?

The tool provided a basic history of the changes to the requirement in the history view of the requirement. In the history view, the names of the users who
modified the requirements and their roles were stored with the change record. It also furnished a side-by-side comparison of the two most recent versions of the requirement. Thus, the tool gave awareness information for this element √

**Use case 6: Examining Requirements**

*Primary Actor: Business analyst and software developer*

**Needs 6.1**

*Pre-condition: Several different stakeholders have modified the project requirements.*

*Goal: The software developer is trying to find out who was working on implementing the requirement before the change, so he or she uses the requirements tool to find who else modified the requirement.*

The developer logged in to the tool. The developer then had the ability to check the history properties of the requirement and see who had modified it. The developer also had the option to review the discussion properties of the requirement and see who had been discussing the requirement.

**Check Use Case 6 Results**

**Check 6.1**

*Did the tool record the name and role of the stakeholder that requested the change as well as an explanation regarding reasons for the change?*

The tool asked for comments whenever a change was made; however, they were not mandatory. In addition, some comments might not be meaningful. Thus, the tool provided some awareness information for this element.

**Check 6.2**

*Did the tool provide the names and roles of the stakeholders who viewed the requirement?*
The tool did not record readership history. Thus, the tool furnished no awareness information for this element.

Check 6.3

Did the tool record an explanation regarding reasons for changes?

\( \checkmark = \) The tool enforces mandatory explicit comments from users making the changes.

4.3.1 Summary

CaliberRM provided at least limited awareness support for many of the views of the requirements change awareness framework. Table 4.4 gives a summary in schematic form. The tool’s major deficiencies appeared in failure to supply workspace location history and readership history. We also found that the tool did not offer support for awareness in case of a requirement deletion.

Although CaliberRM granted limited support for many of the other elements on the awareness framework, it took a rudimentary approach to identifying those requiring notification of changes. The tool provided detailed change affect assessments of requirements, but lacked the ability to supply information to the owners of the requirements about changes that could affect their components of the project.

One of the tools that we found to be particularly useful was the ability of CaliberRM to compare revisions of requirements. This allowed us to follow the sequence of modifications of requirements. This feature was particularly useful when a change did not have adequate commentary.
<table>
<thead>
<tr>
<th>Category of Questions</th>
<th>Information Elements</th>
<th>Use Case Check Coverage</th>
<th>Use Case Needs Coverage</th>
<th>CaliberRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where?</td>
<td>Geographical location history</td>
<td>Check 1.1, 2.1, 3.1</td>
<td>Use cases 1-3</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Workspace location history</td>
<td>Check 1.2, 2.2, 3.2</td>
<td>Use cases 1-3</td>
<td>x</td>
</tr>
<tr>
<td>Who?</td>
<td>Change affect history</td>
<td>Check 1.3, 2.3, 3.3</td>
<td>Use cases 1-3</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
<td>Check 1.4, 2.4, 3.4</td>
<td>Use cases 1-3</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
<td>Check 1.5, 2.5, 3.5</td>
<td>Use cases 1-3</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
<td>Check 6.2</td>
<td>Use case 6</td>
<td>x</td>
</tr>
<tr>
<td>What?</td>
<td>Change history</td>
<td>Check 1.6, 2.6, 3.6</td>
<td>Use cases 1-3</td>
<td>~</td>
</tr>
<tr>
<td>How?</td>
<td>Outcome history</td>
<td>Check 4.1</td>
<td>Use case 4</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Process history</td>
<td>Check 6.1</td>
<td>Use case 6</td>
<td>~</td>
</tr>
<tr>
<td>When?</td>
<td>Event history</td>
<td>Check 5.1</td>
<td>Use case 5</td>
<td>√</td>
</tr>
<tr>
<td>Why?</td>
<td>Motivational history</td>
<td>Check 6.3</td>
<td>Use case 6</td>
<td>√</td>
</tr>
</tbody>
</table>

Table 4-3 CaliberRM Use Case Summary

√ = the tool provides awareness information for this element
~ = the tool provides some awareness information for this element
x = The tool does not provide awareness information for this element
4.3.2 Tool Evaluation: RequisitePro

RequisitePro is available from IBM and provides requirements support to users through a windows client or through a web interface. The web interface and the installable client allow users to add, delete, view, or modify requirements in the repository. It permits users to define customer requirements types with user defined attributes in a fashion similar to that of CalaberRM. Some of the common attribute types are priority, status, and rationale. The ability to customize requirements definitions is an important aspect of requirements management.

RequisitePro also incorporates three ways to view the requirements: an attribute matrix view, a traceability matrix view, and a tree hierarchy view. The attribute matrix view is a table with requirements as rows and attributes as columns. Similar to the table views in CaliberRM, this provides an easy-to-read overview of requirements and their characteristics. The traceability matrix shows the relationships between different requirements. An arrow symbol at their shared index on the matrix indicates requirements that have traceability links between them For example, if Requirement A has a traceability link to Requirement B, then in the matrix at positions (A,B) a symbol of an arrow appears pointing from the row A to the column B. The tree is a hierarchical representation of the requirements and their relationships to other requirements. This is a useful view when trying to understand which requirements are at the core of a project; for example, the root requirements in the requirements tree could have the potential to affect many other child requirements. The requirements tree view is a powerful grouping view of requirements. RequisitePro also allows users to create queries to view custom groups of requirements.

RequisitePro permits users to discuss requirements. It has a feature that allows users to submit comments about individual requirements or the entire project. The tool integrates the discussion feature into an e-mail system that will provide users with e-mail notification on commented issues.
Pre-Use-Case Testing Notes

Setting up RequisitePro is much more challenging than setting up CaliberRM. RequisitePro has a more complicated licensing scheme and requires other software packages such as MS Word and Access. We conducted this test on Rational RequisitePro 2003.06 running on Microsoft Windows Server 2003, on 18 December 2007.

On installation of the application, the user can create a project using a template or create a new baseline template. In our case, we used the templates packaged with the software. Figure 4-22 shows how to set up a new project.

![RequisitePro Baseline Manager](image)

Figure 4-22 RequisitePro: Creating a Project

Use Case 1: Adding Requirements

*Primary Actor: Project manager (or business analyst or developer)*

*Needs 1.1*

*Pre-condition: A new requirement has been identified.*
Goal: The project manager needs to update the project documentation with a new requirement, so he or she creates a new requirement using the requirements management tool.

The first action is for the project manager to log into the client application. The users do not need definition in the project specification before accessing the application. This means that no need for authentication occurs in this particular instance. Figure 4-23 shows logging into the application.

![Figure 4-23 RequisitePro: Logging in as the Program Manager](image)

It was then necessary to decide where the requirement fits into the project. In this case, the new requirement is a product feature. The project manager then selects the folder where the features are located and selects New Requirement, as per Figure 4-24.
The application then allows the user to customize the specifics of the requirements, such as its name and some associated description, as shown in Figure 4-25.
Figure 4.25 RequisitePro: Adding Properties to a Requirement

The project manager then needs to see the attributes of the requirements, such as priority and status. The project manager can set different attributes during the base line creation for the project. Figure 4-26 indicates the setting of requirement attributes.
The next step is to set the traceability links to and from the new requirement. In this case, we selected that the requirement is traceable from two other requirements and has traceability links to one other requirement. Figure 4-27 indicates how to do this.
The tool also has an option to provide hierarchy information to the requirement. The user has the option to set the requirement as a child requirement of another requirement or set the requirement to be the parent of another requirement. Figure 4-28 demonstrates creating relationships between requirements.

![Figure 4-28 RequisitePro: Creating Relationships between Requirements](image)

**Check Use Case 1 Results**

**Check 1.1**

*Did the tool record the geographic location of the project manager when he or she made the change?*

RequisitePro has a loose concept of stakeholders and roles. The application did not distinguish users from each other with respect to geographic location or any other factors for use case 1-3. Thus, the tool did not provide awareness information for this element.

**Check 1.2**

*Did the tool indicate where in the workspace (which elements of the documents) the changes were made?*
For use cases 1 and 2, the tool had the ability to link to design documents or elements in the database. However, RequisitePro implementation had some usability issues that made it difficult for users to access the features. However, if the user is using the “auto suspect” feature, the tool will mark the traceability link between the requirements with a red mark indicating a change the requirements. For use case 2, when a requirement was deleted no record was stored in the project reports or project history. This means that the tool did not store any information about where in the workspace the deletion occurred. Thus, the tool provided some awareness information for this element.

Check 1.3

Did the tool provide the name and the role of those affected by this change?

For use cases 1 and 3, RequisitePro had some support for providing automatic awareness of changes to users. A manual method permits users to stay abreast of changes by allowing them to subscribe to a requirement. This allows users to receive an e-mail notification whenever a revision to a requirement occurs. Several other tools allow users to visualize potential affects from changes to other requirements on their requirements.

The traceability matrix can show how elements of a project relate, see Figure 4-29. A user could examine if a change would affect a requirement for which they are responsible. For use case 2, when a requirement was deleted no record of it was stored in the project reports and history. However, the tool did clean up the project repository and removed all the traceability links to and from the deleted object. Hence, the tool provided some awareness information for this element.
Check 1.4

Did the tool record the names and roles of those involved in the interactions that caused the change?

For use cases 1 and 3, the tool provided a revision history that allowed other users to trace requirements modifications and see a description of how the requirements changed, see Figure 4-30. The tool did not show any synchronous interactions and recorded only the name of the stakeholder that updated the requirement, omitting the names of others involved in the requirement changes. For use case 2, when a requirement was deleted no record of it was stored in the project reports and history. This means that the tool did not store any information about the interactions involved in the deletion of the requirement. This, the tool did not grant awareness information for this element.
Check 1.5

**Did the tool record the name and role information of the person who made the change to the requirement?**

For use cases 1 and 3, the tool did not display any information about the role of the user. The tool did not store any information when the requirement was deleted that could provide support for use case 2. Thus, the tool offered some awareness information for this element.

Check 1.6

**Did the tool record information about all the changes a user made?**

For use cases 1 and 3, the tool showed the changes that the user made, but not a list of all the changes. The tool did not store any information when the requirement was deleted that could support use case 2. Thus, the tool did not provide awareness information for this element.

**Use case 4: Examining Requirements**

*Primary Actor: Business analyst and software developer*

**Needs 4.1**

*Pre-condition: The project has changed status.*

*Goal: The business analyst needs to know the status of the project, so he or she uses the tool to examine the current state of requirements in the project to ensure that requirements have been satisfied.*
Initially, the business analyst will log into the tool. As shown in Figure 4-31, the business analyst could check the status of the requirements by selecting the requirement for the project hierarchy and viewing the details of the requirement. In this case, the analyst will see the Validated status of this requirement.

![Figure 4-31 RequisitePro: Viewing the Status of a Requirement](image)

The business analyst could also access the reports that show the mapping from requirements to use cases. This grants the analyst a high level view of the project requirements coverage of user needs, see Figure 4-32.
Check Use Case 4 Results

Check 4.1

`Did the tool provide outcome histories and can the business analyst check each of the requirements in the project and confirm their coverage by components of the implementation?`

The tool yields detailed outcome histories in the form of status flags and the business analyst can map each requirement to different artefacts and users. Thus, the tool supplies awareness information for this element.

**Use case 5: Examining Requirements**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>UC2: Check Order Status</th>
<th>UC3: Track Packages</th>
<th>UC4: Search by Selected...</th>
<th>UC5: Possiblle New Account</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FEAT1: Secure payment</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure payment method</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEAT2: Easy browsing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy browsing for available titles</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEAT4: Ability to check...</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ability to check the status of an order</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEAT8: User registration...</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shoppers should be able to register once for all...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEAT9: Shipping Status</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shoppers should be able to track any package that has...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEAT16: Andrews New...</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This requirement will allow our product to have the ability to...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Primary Actor: Business analyst and software developer

Needs 5.1

Pre-condition: Modifications to requirements have occurred and the software developers informed of the changes.

Goal: The software developers want to review the requirements because they have information regarding recent changes to the requirements; they use the requirements tool to examine changes.

Initially the developer will log in to the application. The developer can then review the history of the requirement by selecting the requirements history property. If a requirement changed, the tool will use the “auto suspect” feature to mark the traceability link between the requirements with a red mark indicating a change in the requirements. The developer can then select the requirement and review the revision records. Figure 4-33 depicts examining the revisions of the requirement.

![Figure 4-33 RequisitePro: Reviewing Revision Records](image)

Check Use Case 5 Results

Check 5.1

Did the tool record the history of requirements changes; that is, does the tool show the evolution of the requirement and all of the different revisions to the
requirement? Did the tool also indicate the name and role of the stakeholder that revised the requirement?

The tool furnished a basic history of the changes to the requirement in the history view of the requirement. In the history view, the names of the users that modified the requirements were stored with the change record. Thus, the tool provided some awareness information for this element.

Use Case 6: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 6.1

Pre-condition: Several different stakeholders have modified the project requirements.

Goal: The software developer wants to know who was working on implementing the requirement before the change, so he or she uses the requirements tool to find who else modified the requirement.

To start, the developer would login to the tool. The developer could then check the history properties of the requirement and see who had modified it. The developer also has the option to review the discussion properties of the requirement and see who had discussed the requirement.

Check Use Case 6 Results

Check 6.1

Did the tool record the name and role of the stakeholder that requested the change as well as an explanation of why the change was made?

The tool recorded information about the change, such as the user name, time of change, and the fields changed. No fields specifically for comments on the changes exist. Thus, the tool does not provide awareness information for this element.

Check 6.2
Did the tool provide the names and roles of the stakeholders who viewed the requirement?

The tool did not record readership history. Thus, the tool did not give any awareness information for this element.

Check 6.3

Did the tool record an explanation regarding why the change was made?

The tool provided optional comments from users making the changes.

4.3.3 Summary

RequisitePro was particularly weak on providing information about the stakeholders of the project. It does not use any concept of roles or individual user profiles. Since the tool was weak on providing information about users, it was difficult to provide awareness information about who was making changes to projects requirements.

RequisitePro did not track how requirements were changed. It did support mapping requirements defined in the tool for artefacts such as sections of a document. However, this support was difficult to use and did not extend to other artefacts, such as diagrams or source code.

The tool did have two primary awareness features: the auto suspect feature and the revision subscription feature. These features provided users with awareness of changes to project requirements. Unfortunately, the auto suspect feature did not seem to identify hazardous changes: it flagged every requirement that changed as a potential issue. The revision subscription feature was useful, but, unfortunately, required a per-requirement manual subscription to activate, which is somewhat cumbersome. Please see the use case summary in Table 4-5.
<table>
<thead>
<tr>
<th>Category of Questions</th>
<th>Information Elements</th>
<th>Use Case Check Coverage</th>
<th>Use Case Needs Coverage</th>
<th>CaliberRM</th>
<th>RequisitePro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where?</td>
<td>Geographical location history</td>
<td>Check 1.1, 2.1, 3.1</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Workspace location history</td>
<td>Check 1.2, 2.2, 3.2</td>
<td>Use cases 1-3</td>
<td>x</td>
<td>~</td>
</tr>
<tr>
<td>Who?</td>
<td>Change affect history</td>
<td>Check 1.3, 2.3, 3.3</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
<td>Check 1.4, 2.4, 3.4</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
<td>Check 1.5, 2.5, 3.5</td>
<td>Use cases 1-3</td>
<td>√</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
<td>Check 6.2</td>
<td>Use cases 6</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>What?</td>
<td>Change history</td>
<td>Check 1.6, 2.6, 3.6</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>How?</td>
<td>Outcome history</td>
<td>Check 4.1</td>
<td>Use case 4</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Process history</td>
<td>Check 6.1</td>
<td>Use case 6</td>
<td>~</td>
<td>x</td>
</tr>
<tr>
<td>When?</td>
<td>Event history</td>
<td>Check 5.1</td>
<td>Use case 5</td>
<td>√</td>
<td>~</td>
</tr>
<tr>
<td>Why?</td>
<td>Motivational history</td>
<td>Check 6.3</td>
<td>Use case 6</td>
<td>√</td>
<td>~</td>
</tr>
</tbody>
</table>

**Table 4-4 RequisitePro Use Case Summary**

√ = the tool provides awareness information for this element  
~ = the tool provides some awareness information for this element.  
x = the tool does not provide awareness information for this element
4.3.4 Tool Evaluation: DOORS

DOORS is available from Telelogic for management of requirements. DOORS provides a hierarchical view in the user interface where requirements are displayed with associated attributes. Other objects, such as images and documents, can be stored in the folder hierarchy and, therefore, associated with requirements and other artefacts. Attributes can be user-defined. DOORS supports the notion of dependency linking between requirements and artefacts. To link one requirement to another, the user simply drags and drops one requirement icon onto the desired link. The DOORS software also has a change proposal system to suggest modifications. Finally, another interesting aspect of DOORS is that it has the concept of users. The software allows the user to create customized user profiles in the project modules that allow a highly granular read and write access to the project.

Pre-Use-Case Testing Notes

Configuring DOORS was much more challenging than setting up CaliberRM. This test was conducted on Telelogic DOORS 8.3 with the database running on Microsoft Windows Server 2003, and the client on Windows XP. The test occurred 4 January 2008. It was necessary either to create a new project on the server or to use an existing template. For the purposes of this evaluation we used an existing template packaged with the tool. DOORS allows the user to customize a project using a wizard application. The user must define the project type, in this case, a software project. Figure 4-34 shows the creating on a new project.
Using the wizard, a user can define the attributes for the requirements. The user may add different sets of attributes to different modules and create custom attributes. Figure 4-35 demonstrates adding new attributes to a project. In the next step, the user can set up some visualization and analysis views, as shown in Figure 4-36.
Figure 4-35 DOORS: Defining Requirement Attributes
The tool also allows the user to create individual user accounts with some limited information such as contact information and location. Figure 4-37 shows adding a new user and role. Each new user can have specific permission within the project, defined from a template of customized configurations. Figure 4-38 shows how to add detail about the users.
Figure 4-37 DOORS: Creating User Accounts

Figure 4-38 DOORS: Customizing User Permissions
Use Case 1: Adding Requirements

*Primary Actor: Project manager (or business analyst or developer)*

**Needs 1.1**

*Pre-condition: A new requirement has been identified.*

*Goal: The project manager needs to update the project documentation with a new requirement, so he or she creates a new requirement using the requirements management tool.*

The project manager logged in to the client. See Figure 4-39. After the project manager logs in to the client, he or she will see a list of all projects in the database.

![Login - D...](image)

**Figure 4-39 DOORS: Logging in as the Project Manager**

The user then selects the project in which he or she is interested, adding a new requirement, and then selects the requirements folder. He or she then selects the type of requirement to add. In this case, the project manager will add a “User Requirement.” See Figure 4-40.
Once the user has opened the “User Requirements” document, he or she sees a list of all the existing requirements. The user can then add a requirement as a sub-requirement to another requirement, or he or she can create a new requirement at the base level. See Figure 4-41.
The user then selects the section of the document into which they want to insert the requirement and selects the inset object option. Figure 4-42 demonstrated moving a requirement. Next, the project manager will add a description of the requirement. Figure 4-43 shows how to add detail to the requirement.
Figure 4-42 DOORS: Selecting Where to Insert a Requirement
Figure 4-43 DOORS: Adding a Description to a Requirement

The project manager can then add access controls to the requirement and can specify who can read, modify, or create changes to the requirement. See Figure 4-44.....
The user can then set such attributes of the requirement as the status. The user can also customize other attributes for the requirement. See Figure 4-45
The user can then set traceability links between internal or external project requirements. He or she may add an external requirements URL to the requirement link profile or drag an internal requirement icon onto the new requirement icon. See Figure 4-46.
The final step is to save the changes and the tool will add the requirement to the database. On saving the changes to the document the coloured flags, will turn from red to yellow. See Figure 4-47 and Figure 4-48.

Check Use Case 1 Results

Check 1.1
Did the tool record the geographic location of the project manager when he or she made the change?

For use cases 1-3, although the tool provided a user profile view with a location field, the tool did not offer that information to users. The only person who can see the user location is the application administrator who can view the full user profile. Thus, the tool does not supply awareness information for this element.

Check 1.2

Did the tool indicate where in the workspace (which elements of the documents) the changes occurred?

For use cases 1 and 3, the tool provided a colour coding of requirements modified by users. Blue indicates that the requirements are part of the project baseline and yellow indicates an addition or modification to the requirement. See Figure 4-48. This granted partial information regarding when in the workspace a change happened, but required the user to be in the module where the change has occurred. In addition, when a user positioned his or her mouse cursor over the changed requirement, the time and the username of the user who made the change was displayed.

![Figure 4-48 DOORS: Saving Changes to a Requirement](image)

The tool also provided a graphical view of the module. The graphical view allowed the user to see the requirement within the modal and visualize the project hierarchies. See Figure 4-49.
For use case 2 (deleting a requirement), the user set the application to show the deleted requirements. This allowed the user to see all the deleted requirements in red. This view showed where in the module the deletions happened, but did not provide support for showing changes across modules. See Figure 4-50. Thus, the tool did not support awareness information for this element.
Figure 4-50 DOORS: Viewing Deleted Requirements
Check 1.3

*Did the tool provide the name and the role of those affected by this change?*

For use cases 1 and 3, the properties section of the requirement had a history field where users could see who added the requirement. They could also see what other requirements were linked to this requirement and the users associated with that requirement. This was a manual process. In general, no visualization displayed all of the users affected by changing the requirements. For use case 2, the tool furnished an error message when trying to delete the requirement. This means that before a user can delete a requirement, he or she must unlink it from other dependent modules. See Figure 4-51. This granted partial awareness of other modules affected by the change.

Unfortunately, when the user unlinked the module and deleted it, he could no longer ascertain those affected by the change. Thus, the tool did not provide awareness information for this element.
Check 1.4

Did the tool record the names and roles of those involved in the interactions that caused the change?

For use cases 1 and 3, the tool provided two ways to identify who added the requirement. The first was to move the mouse over a yellow colour-coded requirement. The second way involved opening the requirement and examining the properties. The history tab listed all the users who made changes to the requirements. See Figure 4-52. However, the tool did not show any information about other interactions. For use case 3, the tool yielded the change history information about the requirement. This information indicated the name of the user who deleted the requirement. The tool did not offer any collaborative information about the change. Thus, the tool granted some awareness information for this element.
Figure 4-52 DOORS: Viewing the Change History

Check 1.5

*Did the tool record the name and role information about the person who made the change to the requirement?*

For use cases 1 to 3, DOORS offered two ways to identify who added the requirement. The first required moving the mouse over a yellow colour-coded requirement. The second way involved opening the requirement and examining the properties. The history tab showed all the users who made changes to the requirements. The tool did not show the role of the user. Thus, the tool provided some awareness information for this element.
Check 1.6

Did the tool record information about all the changes a user made?

For all use cases 1 to 3, the tool showed the changes that the user made, but not a list of all the changes. Thus, the tool did not provide awareness information for this element.

Use Case 4: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 4.1

Pre-condition: The project has changed status.

Goal: The business analyst needs to know the status of the project, so he or she uses the tool to examine the current state of requirements in the project and to ensure that requirements have been satisfied.

First, the business analyst will log in to the application, as in Figure 4-53. Then he or she will select to add a new column to the modules view and add the status attribute. The new column will provide the status of each element of the project. See Figure 4-54.
Figure 4-53 DOORS: Choosing to View the Status of a Requirement
Figure 4-54 DOORS: Viewing the Status of Each Requirement

It is also possible for the user to view the requirements in other view modes, such as the risk assessment mode. See Figure 4-55. This view mode allows the user to see a risk value associated with each requirement.
Check Use Case 4 Results

Check 4.1

Did the tool provide outcome histories and can the business analyst check each of the requirements in the project and confirm their coverage by components of the implementation?
The tool gave status information to the user. The tool also allowed for customizing views and attributes so users can run reports on different types of project metrics. Thus, the tool provided awareness information for this element.

**Use Case 5: Examining Requirements**

*Primary Actor: Business analyst and software developer*

**Needs 5.1**

*Pre-condition: Requirements have changed and the software developers are aware of the changes.*

*Goal: The software developers want to review the requirements because they have been informed of recent changes to the requirements. They use the requirements tool to examine modifications to the requirement.*

The software developer then logs in to the tool. He or she then identifies the requirement under scrutiny by selecting the properties option for the requirement. Using the history tab, the developer can review all the changes to the requirement. See Figure 4-56. The tool also provides a view that shows modifications to the requirement and who was responsible.
Figure 4-56 DOORS: Reviewing Changes to a Requirement

Check Use Case 5 Results
Check 5.1

Did the tool record the history of requirement changes; that is, did the tool show the evolution of the requirement and all of the different revisions to the requirement? Did the tool also provide the name and role of the stakeholder that revised the requirement?

The tool indicated the changes to the requirement and who made them. However, the tool did not provide the role of the user that made the changes. Thus, the tool provided some awareness information for this element.

Use Case 6: Examining Requirements

Primary Actor: Business analyst and software developer

Needs 6.1

Pre-condition: Several different stakeholders have modified the project requirements.

Goal: The software developer is trying to ascertain who was implementing the requirement before the change, so he or she uses the requirements tool to find who else modified the requirement.

The software developer then logs in to the tool. He or she selects the requirement and reviews the details on the history log.

Check Use Case 6 Results

Check 6.1

Did the tool record the name and role of the stakeholder that requested the change as well as an explanation of the change?

The tool provided the name of the users who made changes to the requirements. Where requirements had a change proposal submitted, a motivational comment for change was present. However, in most cases the tool did not document well motivations for changes. Thus, the tool provided some awareness information for this element.
Check 6.2

Did the tool provide the names and roles of stakeholders who viewed the requirement?

The tool did not record readership history. The tool did not provide awareness information for this element.

Check 6.3

Did the tool record an explanation regarding the change made?

The tool did not enforce mandatory explicit comments from users making changes.

4.3.5 Summary

DOORS was particularly weak in providing information about project stakeholders. Although user management was very good from a security perspective, it was difficult to determine the physical location of users or their roles in the project. The tool also offered very little information about why changes were made; few places were available to add comments about changes and the tool had no built-in communication tools, such as the bulletin boards found in other products. As well, DOORS is a complex software program and new users might find it difficult to use the application without training.

DOORS did support event history, but without more detailed role information the event history was not as useful as it might have been. DOORS did not make users submit comments when making changes to the project. Comments were required only for such operations as suggesting a change proposal. This made it difficult to assess the history of the project. The tool did grant some support for identifying changes to requirements and their affects in the workspace, but the tools were rudimentary and required considerable manual effort since no way existed to automate this type of reporting.
The DOORS application provided several beneficial features, including the ability to view and define project attributes. The tool was extremely flexible and allowed users to customize their projects. More specifically, DOORS offered the best support for deleted requirements: the software will not remove the requirements and their associated data when deleted unless the administrator also purges the requirement from the database. This is useful when understanding changes to a project. See Table 4-6 for a use case summary.
### Table 4-5 DOORS Use Case Summary

<table>
<thead>
<tr>
<th>Category of Questions</th>
<th>Information Elements</th>
<th>Use Case Check Coverage</th>
<th>Use Case Needs Coverage</th>
<th>CaliberRM</th>
<th>RequisitePro</th>
<th>Doors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where?</td>
<td>Geographical location history</td>
<td>Check 1.1, 2.1, 3.1</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Workspace location history</td>
<td>Check 1.2, 2.2, 3.2</td>
<td>Use cases 1-3</td>
<td>x</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Who?</td>
<td>Change affect history</td>
<td>Check 1.3, 2.3, 3.3</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Interaction history</td>
<td>Check 1.4, 2.4, 3.4</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Authorship history</td>
<td>Check 1.5, 2.5, 3.5</td>
<td>Use cases 1-3</td>
<td>√</td>
<td>x</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>Readership history</td>
<td>Check 6.2</td>
<td>Use case 6</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>What?</td>
<td>Change history</td>
<td>Check 1.6, 2.6, 3.6</td>
<td>Use cases 1-3</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>How?</td>
<td>Outcome history</td>
<td>Check 4.1</td>
<td>Use case 4</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Process history</td>
<td>Check 6.1</td>
<td>Use case 6</td>
<td>~</td>
<td>x</td>
<td>~</td>
</tr>
<tr>
<td>When?</td>
<td>Event history</td>
<td>Check 5.1</td>
<td>Use case 5</td>
<td>√</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Why?</td>
<td>Motivational history</td>
<td>Check 6.3</td>
<td>Use case 6</td>
<td>√</td>
<td>~</td>
<td>x</td>
</tr>
</tbody>
</table>

√ = the tool provides awareness information for this element
~ = the tool provides some awareness information for this element
x = the tool does not provide awareness information for this element
Chapter 5

Discussion

Here, we review the results of our evaluation and discuss the relationship between the evaluation outcomes and our proposed framework. We will see that one of the tools performed much better than the other two.

5.1 Tool Deficiencies

During our evaluation, we observed several trends identified in the tool evaluation. The tools provided explicit support for some of the elements in the framework and little for others. Interestingly, RequisitePro and DOORS supplied similar support for awareness of requirements changes when evaluated using our framework. All three tools yield explicit support for outcome history, giving some level of awareness information to stakeholders on the status of requirements and how they have changed.

CaliberRM distinguished itself with strong support for roles. The tool provided full support for authorship history, motivational history, and event history. The makers of CaliberRM designed their product with a focus on collaboration and instituted well-defined roles and user identities. Their focus on collaboration and tight concepts of identity allowed them to establish a high level of awareness support for changing requirements. RequisitePro and DOORS, on the other hand, had weaker notions of roles and, therefore, were able to give only limited support for authorship history and event history.

Unfortunately, we also found that the tools were all deficient in four areas. The tools were ineffective in providing information on geographical location history and readership history. They also seemed weak in providing support for
workspace location history, change affect history, interaction history, and process history.

With respect to the geographical location history, none of the tools offered substantial support for any concept of locality of stakeholders. The tools did not fully take into account that stakeholder locations might lie in different physical environments. The tools did not provide information on stakeholder distribution and the environments within which each user operates. As mentioned earlier, geographic distribution negatively affects team cohesion and organizational awareness. When reviewing changes by others in the team, it was difficult to know where the people making the changes were geographically located. This is important when trying to contact them to collaborate on developing the requirements.

When analyzing readership history, we identified that the tools provided little support for identifying who had viewed project requirements or artefacts. Allowing stakeholders to access awareness information about who has reviewed the requirements is useful for providing improved shared understanding of a project. When evaluating the tools from the perspective of a project manager, it would have been very useful to know if the other members of the team had observed that the requirement had changed. This would allow the project manager to ensure that the right people had reviewed the new information.

The tools do offer limited support for awareness information about where changes happen in the workspace, but in all cases it seemed more like an afterthought than a fully developed feature. The tools provide various visual clues about where changes happen in the workspace, such as by colour coding of requirements that have been modified and by using icons to mark requirements with suspect traceability links. Unfortunately, these techniques are useful only if the user makes use of very specific views in the tools. No support appeared for providing an overview of an entire project with various locations.
With respect to change affect history, the tools provided an analysis of the requirements affected by modifying others. The affect assessments were very rudimentary and normally identified only the requirements linked directly with traceability links. No transitive analysis identified requirements affected by changes to adjacent requirements. Perhaps the main weakness of the tools’ abilities to furnish change affect history was that they did not identify the stakeholders affected by the changes. It was possible to manually review the requirements and find the related stakeholders, but to apply this process consistently on a regular basis would be very time consuming.

With respect to interaction history, the tools provided very little to identify who had collaborated on changing requirements. Perhaps this element of our framework was the most ambitious, since it is difficult to identify all interactions and map them to changes in requirements. It is possible that collaborations that lead to requirements changes, based on the circumstances under which they occur, might not be sufficiently documented. For example, a conference call or meeting would be difficult to translate into the tool environment, and the awareness tool cannot, therefore, store and later present that information.

Process history provides incremental information on how the project has changed. The process history was also difficult to ascertain from the tools. Some of the tools incorporate methods for adding comments when modifying requirements, but none of the tools required them. It is also difficult to ensure that when the users do make comments about changes that they are actually meaningful and contain motivational information about the changes.

All tools focused on project artefacts rather than stakeholders. To provide more accurate awareness information, the tools need to supply information about the stakeholders and their activities. The tool that offered the best support for understanding stakeholders was CaliberRM. Nonetheless, the producers of this tool did not go far enough and their concepts of roles seemed like a secondary concern.
5.1.1 Improved Framework

After developing the framework and applying it to evaluate three commercial RE management tools, we discovered that several elements in our evaluation model were not incorporated into the various tools. Geographic location history and readership history have the least support in these tools. This could imply either that these elements are not necessary components of the RE awareness framework or that tool manufacturers have neglected support for these information elements. After reconsidering current literature and, in particular, the work of Tam and Greenberg (2006), we concluded that these are indeed necessary for the framework and believe that these elements provide valuable awareness information that could easily form part of requirements management tools. According to Tam and Greenberg (2006), readership history is particularly important since it furnishes an implicit acceptance of project changes. The fact that these two elements have minimum support is indicative of a current misconception of requirements change awareness. These elements are particularly relative to users, rather than artefacts or requirements, and, hence, seem critical to translating the current requirements management environments into collaborative platforms for stakeholders. Tools for RE should preserve both project documentation and translate project information into pieces of knowledge that can enhance awareness and foster collaboration.

The tools provide inadequate support for workspace location history, change affect history, process history, and interaction history. In the first two cases, it would require minimal effort to include these features in requirements management tools. For example, producers could implement workspace location history in a graphical way for an overview of the workspace and its changes. The overview would allow users to verify the areas of the workspace on which stakeholders have focussed. Change affect history could be improved by providing tools that allow users to see the global effect of requirements changes, not only identifying directly affected artefacts and users, but also projecting the
reverberation of modifications throughout the software development lifecycle. In the case of the interaction history, we realize that it might be impractical to provide detailed records of the group interactions that lead to the change of requirements. This is because some of the interactions would occur face-to-face and not be recorded by electronic means. Thus, we suggest that the element be removed from the awareness framework.

While evaluating the tools, we also realized that our evaluation methodology was not capturing another element. Although some of the tools provided the same awareness information, it seemed that the different tools would present the information in different ways. The way in which the tools presented awareness information seemed to have an affect on the effectiveness. We then asked ourselves, “How are changes to requirements visualized?”

This question brought us back to the awareness visualization framework developed by Storey, Cubranic, and Germán (2005). These authors provided a detailed framework for evaluating tool support for awareness visualization. Specifically, the presentation view of their framework could be incorporated into the framework proposed by this thesis. The view should focus on describing the types of visualizations that support awareness. It should include graphical views of requirements and relationships, customizable views, and annotated views as well as the visual variables used to describe the requirements and related artefacts.

Although the framework seems to be missing a way to include the effectiveness of presenting awareness information, it still captures all the necessary information required to provide awareness. With that in mind, we would not necessarily add awareness visualization as an information element to the framework, but, rather, suggest that our framework be used in conjunction with the one developed by Storey et al. (2005). The development of a requirements change awareness framework has increased our understanding of requirements change awareness, since our framework allowed us to identify differences between traditional RE and awareness enhanced RE. The information
elements of the framework captured these differences. By applying the framework, we were able to identify gaps in the framework that were not taken into consideration during the framework construction.

Some of the issues we faced in our research were to capture a set of use cases aimed at accurately replicating archetypical situations in requirements change awareness. Although our use cases reflect common activities in software projects, they were by no means comprehensive: in the real world projects will vary in RE demands and methodologies. We also faced challenges in performing the use cases with only one evaluator. In reality, project changes arise from different people with differing perspectives and opinions: setting up cases with the complexity of those that arise while dealing with multiple, distinct people was not possible in our evaluation. In future work, we would like to apply the framework to evaluate tools in a real world field study. Applying the framework to evaluate the tools presently used by practitioners would help us validate our framework.

Although changing some components of our methodology, such as using templates, in our future work might change aspects of our conclusion, we feel that our current results would most likely remain consistent with the refinements in the methodology; although changes might take place in some of the evaluation variables, the information flow within each tool would remain the same. That is, increasing the complexity of the project teams and processes would not change the fact that the tools provide awareness information explicitly, implicitly, or not at all.

The other aspect of the framework that could be refined is the granularity with which the evaluation scores each tool on adherence to each information element. We extracted the scoring methodology from the work of Tam and Greenberg (2006), which used a three point rating scale (explicit support, implicit support, or no support). We found that it was common for tools to provide some implicit support for information elements, but that level of support could vary
dramatically within the category itself. This seems problematic since two tools might be classified as providing some implicit support for various information elements and, therefore, might be considered to provide similar support for requirements change awareness. In reality, however, one tool might offer a higher level of support for the element. Although it would be difficult, it is recommended that future work create a better way of scoring each tool against the different elements.

5.1.2 Summary

In this chapter, we have discussed how our framework performed with respect to our tool evaluations. We identified several possible additions and deletions to the proposed awareness framework. In particular, we identified that the interaction history component of the framework should be modified due to the lack of ability to document all interactions that result in requirements modifications. We also suggested that it is important to add the concept of awareness presentation to the framework to allow for describing the types of visual support in providing awareness information.
Chapter 6

Conclusion

As discussed in Chapter One, requirements are the foundation of software projects and directly link to the project design components. A lack of awareness of requirements changes can negatively affect projects. As a result, it is essential that stakeholders have information regarding requirements changes that affect their work. Improving awareness of changes to requirements has the potential to improve a software developer’s decision-making as well as communication between stakeholders.

This thesis furthers our understanding of awareness in RE. This is important because little is known about awareness of requirements, and we lack frameworks for the evaluation of requirements change awareness support in current requirements management tools. Our contributions are twofold: a framework and insights regarding RE tools.

Our framework captures the elements important to the concept of awareness in RE. The framework examines elements such as the nature of changing requirements and their relationships to project roles. The framework is useful for both researchers and practitioners in informing tool design and evaluation since it captures the important pieces of information necessary for providing awareness of changing project requirements.

This study has produced insights from the evaluation of three commercial RE management tools, RequisitePro, DOORS and CaliberRM. Specifically, an evaluation that demonstrated the utility of the framework in evaluating awareness support in RE management tools revealed that the different tools provide a different level of awareness information.
This thesis helps to fill the knowledge gap in understanding the concept of awareness in software development and specifically with respect to awareness of RE. As current requirements management practices rarely promote awareness, this research hinged on the development of a framework from which we could draw tool evaluation results.

The research was divided into three components: a literature survey, framework development, and application and improvement of the framework through a requirements management tool evaluation. Each of the components built upon the work developed in the previous one.

We then developed a framework for discussing requirements change awareness based on the literature reviewed. Following the development of the framework, we iteratively improved it through its application to a process of evaluation of awareness support in three current RE management tools.

6.1 Insights

Awareness support is an important element in RE. It was surprising to find that most of the tools evaluated against our awareness framework did not provide high levels of awareness support, with the exception of CaliberRM. It seems that a focus on developing traceability between requirements and artefacts has excluded work on expanding the collaborative features of requirements management systems.

We also found that developing a framework for requirements change awareness had specific challenges in capturing actions and activities related to changes not explicitly entered into the RE tools. In particular, information related to interaction history was very difficult to capture. Other insights to requirements change awareness relate to the lack of adequate support for awareness visualizations within the RE tools we evaluated. Furthermore, we felt that the next step in improving our understanding of requirements change awareness would be to apply the framework to a field study and evaluate tools in actual use.
6.2 Practical Applications

After evaluating several requirements tools widely used in industry, we identified several deficiencies in the tools with respect to awareness. It is our recommendation to use this work evaluating tools and for tool development. Tool developers can use our framework to guide their design of collaborative tools.

As discussed in the literature review, many practical benefits accrue to improving RE and awareness. Some of the tangible benefits are better management of project scope, schedules, and resources.

6.3 Future Work

We would like to perform several activities in future work. The first is to validate further our requirements change awareness framework. We will do this by applying the framework in the field and observing whether the results are consistent with our laboratory evaluations. We also want to validate the framework by having it examined and gauged by experts.

We would also like to evaluate other RE management tools, especially open source tools. Open source tools allow us to examine whether assigning requirements change awareness features in the tool helps users become more efficient with their work.

As we mentioned in Chapter Two, awareness information can be presented in different ways. The manner of presentation of information seems to have a direct affect on the effectiveness of the tool when providing awareness. We would like to examine this further and incorporate the concept of awareness presentation effectiveness into our framework.

Another direction for future work is the refinement of evaluation scoring for RE tools against the information elements of the framework. Refining the
evaluation scoring could help us prioritize the information elements when evaluating tools and provide more granularity in the evaluation use cases.
References


