A BUSINESS RULE APPROACH TO REQUIREMENTS TRACEABILITY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT NARMANLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2010

Approval of the thesis:

A BUSINESS RULE APPROACH TO REQUIREMENTS TRACEABILITY

submitted by **MURAT NARMANLI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Ali Hikmet Doğru _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Ahmet Coşar _____
Computer Engineering Dept., METU

Assoc. Prof. Ali Hikmet Doğru _____
Computer Engineering Dept., METU

Dr. Cevat Şener _____
Computer Engineering Dept., METU

Dr. Kıvanç Dinçer _____
TÜBİTAK – UEKAE

Dr. Ahmet Tümay _____
TÜBİTAK – UEKAE

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : MURAT NARMANLI

Signature :

ABSTRACT

A BUSINESS RULE APPROACH TO REQUIREMENTS TRACEABILITY

Narmanlı, Murat

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Ali Doğru

September 2010, 67 pages

In this thesis, a requirements traceability model is proposed in order to make efficient and effective *change request impact analysis*. The proposed model is a *requirements – requirements* traceability model. There are several researches regarding software requirements traceability problem. The main problem of these researches is that the proposed solutions can not be applied to software industry with affordable changes. However, current literature begins to see that describing all the software requirements in a huge black box is not so much applicable to today's more dynamic and bigger software projects, especially regarding *change management*. The proposed traceability model tries to be a solution to these problems. *Change requests* and *business rules* are two important and popular terms for today's software industry. The traceability model consists of three types of software requirements: *data definitions*, *business rules* and *use cases*. The traceability model proposes *bidirectional traces* between these types. Data definitions, business rules and use cases are related to each other and they all should be seen as parts of a software system which should work together to make the software system work properly. Empirical investigation is made on a real industrial software project. These types were configured in order to match to the project specific needs in a reconfigurable way. Experimental results show that the traceability model has an acceptable degree of correctness.

Keywords: Requirements Traceability, Change Management, Change Requests, Business Rules, Impact Analysis

ÖZ

GEREKSİNİM İZLENEBİLİRLİĞİNE İŞ KURALI YAKLAŞIMI

Narmanlı, Murat

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ali Doğru

Eylül 2010, 67 sayfa

Bu tezde, verimli ve etkili bir şekilde *değişiklik isteği etki analizi* yapabilmek için bir gereksinim izlenebilirlik modeli sunulmuştur. Sunulan model bir *gereksinim – gereksinim* izlenebilirlik modelidir. Yazılım gereksinimleri izlenebilirlik problemi hakkında bir çok araştırma bulunmaktadır. Bu araştırmaların ortak problemi, sunulan çözümlerin yazılım endüstrisine karşılanabilir değişikliklerle uygulanamamasıdır. Fakat, güncel literatür bütün yazılım gereksinimlerinin büyük, siyah bir kutu içerisinde tanımlanmasının bugünün daha dinamik ve büyük çaplı yazılım projelerine, özellikle *değişiklik yönetimi* yönünden uygun olmadığını görmeye başlamıştır. Sunulan izlenebilirlik modeli bu problemlere bir çözüm getirmeye çalışmaktadır. *Değişiklik istekleri* ve *iş kuralları* bugünün yazılım endüstrisi için önemli kavramlardır. İzlenebilirlik modeli üç çeşit yazılım gereksinimden oluşmaktadır: *Veri tanımları*, *iş kuralları* ve *kullanım durumları*. İzlenebilirlik modeli bu türler arasında *çift yönlü izlenebilirlikler* önermektedir. Veri tanımları, iş kuralları ve kullanım durumları birbirleriyle ilişkilidir ve bir yazılım sisteminin düzgün şekilde işleyebilmesi için birlikte çalışması gereken yazılım sistemi parçaları olarak görülmelidirler. Deneysel araştırma gerçek bir endüstriyel yazılım projesi üzerinde yapılmıştır. Bu türler projeye özel ihtiyaçlar için tekrar düzenlenebilir şekilde ayarlanmıştır. Deneysel sonuçlar izlenebilirlik modelinin kabul edilebilir bir oranda doğruluk seviyesine sahip olduğunu göstermektedir.

Anahtar Kelimeler: Gereksinim İzlenebilirliği, Değişiklik Yönetimi, Değişiklik İstekleri, İş Kuralları, Etki Analizi

To my family

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and profound respect to my supervisor Assoc. Prof. Ali Hikmet Doğru for his expert guidance and suggestions, positive approach throughout my master study and his efforts and patience during supervision of the thesis.

I would like to especially thank the jury members, Assoc. Prof. Ahmet Coşar, Dr. Cevat Şener, Dr. Kıvanç Dinçer and Dr. Ahmet Tümay for their contributions to my thesis.

I wish to thank TÜBİTAK UEKAE / G222, especially to the members of İKİS project for their support and ideas which helped me to complete my thesis.

I am grateful to my family who loved and supported me throughout my life. They made me who I am.

Finally, I want to specially thank Gülseren Gülay for her endless love, understanding and sensibility.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Change request and *business rule* are two important and popular terms for today's software industry. Change request is a method for indicating a request for changing a part of the whole software formally. Business rule is defined as "*A policy your software must satisfy. Business rules are what a functional requirement "knows," the controls and guidelines that are fulfilled by the functional requirement. An operating principle or policy of your organization.*" [36].

Change requests are important communication links between stakeholders of a software project. Change requests can come from all the stakeholders including developers and end-users. Each change request has an impact on the current configuration of the software that should be analyzed and handled carefully. In order to make impact analysis in a well-defined and clear way, traceability must exist between every item of software related to each other.

There are several commercial requirement management tools at the market supporting requirements traceability; researches about effective traceability methods as well. The main problem of these researches is that they can not be applied to industry with affordable changes. Vendors do not show enough attention to traceability models of different projects and theoretic traceability methods.

Customers and software development team can not speak the same language. Developers like to talk in technical terms while customers like to talk in daily language. A customer can not be expected to tell the development team everything that is enough to build software requirements specification (SRS).

Business rules are a bridge between customers and the software which developers build for them. Business is expressed by business rules [21]. The main information that customers,

especially end-users of a system can give is business rules of the process. Business rules are human-readable, easily-maintainable, effectively-traceable concepts.

Customer is the core of today's competitive software industry; hence development team has to understand customers and re-tell what customer said in a well-defined and comprehensible way. Under these circumstances, business rules have a very important and dynamic role as they can be used in all the phases of a software project like requirements development and management, technical solution, implementation, verification and validation. They are very important interfaces for change requests. That is why change requests and business rules should be handled together.

1.1 MOTIVATION

There are many researches about tools being used for requirements traceability; therefore making a review of current tool support for requirements traceability should not be aimed at. Traceability models and methods should be concentrated on in order to build the software that customers want in an effective way and make verification-validation phases easier by expressing software requirements in a well-defined and comprehensible way. Neither expecting commercial firms to build what development teams exactly want, nor using the capabilities of existing tools without any methodology is the right way. Establishing and maintaining a usable requirements traceability model should be aimed at; because Spanoudakis and Zisman [1] states that

> *However, despite its importance and the work resulted from numerous years of research, empirical studies of traceability needs and practices in industrial organizations have indicated that traceability support is not always satisfactory. As a result, traceability is rarely established in existing industrial settings.*

A traceability model should be simple [29] as traceability is a tool, not an aim. Hence, not a complex solution, but a simple and an applicable one was looked for. Industry always demands simple and quickly applicable solutions which it can apply with their existing personnel and tools. They do not want to spend extra money for trainings of new tools. Industry's first aim is to satisfy customers' needs with a good quality degree of software which has a fast and cheap solution; because they are always in a hurry to catch-up with the deadlines while customers usually expect them to read their minds and build what they exactly want.

Requirements traceability provides estimation support for software projects to see how big the system is, what is needed to build it regarding human effort and technology. But traceability is not enough; effective measurement and analysis repository is also essential. If the model and understanding of traceability management are not standardized, moreover not integrated to internal processes; creating and maintaining traceability links becomes a waste of time.

Software projects have vertical links [25] in the design, but usually not in the software requirements. Structure of the software requirements should be usable and helpful to create the software design. In order to take advantage of the software requirements and build a system matching to what customer wanted, there should be links in the software requirements as in the design. This requires a traceability model in the software requirements.

Update of the software requirements is very important for the reason that the main condition for the acceptance of the software by the customer is conformance to the software requirements specification. Conversely, update of the software design is not as critical as requirements; therefore software design descriptions are generally updated from release to release of the software. Implementation tells about the design a lot and the design can be formed by using reverse engineering of the code by several tools, as well. Moreover, developers have a well understanding and memory about the design but this is not valid for the requirements. First of all, impact analysis should be done when a change request come. To achieve this analysis, traceability model is needed in the software requirements.

Customer requirements must be caught near to their sources in an atomic way. This makes understanding and asking questions about them easy. Most of the customer requirements are in shape of business rules when coming out from the source. Hence, the best way of collecting customer requirements is to express them in business rules and trace them as soon as possible. This is also a good starting point for creating software requirements. At this point, defining a requirements traceability model before collecting requirements becomes an important factor to have meaningful and traceable software requirements. Ambler [36] states that

> *Part of managing complexity is being able to respond quickly when your environment changes. Business needs change and you need to be able to react quickly to those*

changes. … By implementing complicated concepts and business rules in objects, you can build complex systems much more quickly.

Business rules are identified in the normal course of requirements gathering and analysis. While you are use case and domain modeling, you will often identify business rules.

Development teams do not care about business rules at the beginning of the software project in general. Hence, at the end of a software project, a problem can be found even at acceptance tests. While collaborating with the customer at the beginning of the requirement analysis; if enough effort does not exist or business rules are not seen valuable enough for formally indicating; in later phases, business rules come as change requests and rework in spite of customer requirements.

There are a few business rules communities [38], [42] and their numbers are increasing as the industry understands their value. They concentrate on the business rules and emphasize how business rules make ease of their life.

One of the approaches for software requirements is use-case approach. It is widely accepted and used by the industry. Use-case steps aims interaction between the end-user and the system. A use case describes "who" can do "what" with the system in question [39].

Use-cases and business rules are different in nature. Use-cases have several advantages, but it is not possible to express all the business by using pure use-case approach. Most of the sentences that customers say are business rules, not steps in a use case. Business rules look at the system from viewpoint of business, the core of the system. Therefore, it is crucial to give enough attention to collect business rules in a formal way. It is a common mistake to put business rules as notes into use-cases [21]. By this way software design and implementation becomes complex and highly coupled. That is why business-rule engine approach gains attention and many of them are evolving [43], [44], [45].

Data definitions should be mentioned as software requirements. They are at the core, they are metadata. They should be traced and synchronized with other software requirements by a requirement tool, not as excel files, etc. Use-cases, business rules and data definitions are related to each other and they all should be seen as parts of a system that should work together to make the system work properly. Business rules are open to change and this

makes change requests and business rules good friends. Therefore, it is a good idea to build a traceability model with these software requirement types and make it business rule based.

1.2 THESIS ORGANIZATION

The organization of the thesis work is as follows: In *Chapter 2*, definition of related concepts, the history of the requirements traceability problem and the related literature survey is presented. In *Chapter 3*, the background and the proposed requirement traceability model itself is described in detail. In *Chapter 4*, related experimental study made method of the study, evaluations and some other comments are given. *Chapter 5* concludes with the main results of the thesis work, reasons about the results, in addition some possible enhancements and future works.

CHAPTER 2

BACKGROUND INFORMATION AND RELATED WORK

2.1 INFORMATIVE PART

This part gives information about widely accepted general terms and best practices; moreover have a look at *change management* and *requirements traceability* concepts from CMMI [37] point of view through "*Requirements Engineering"*, "*Requirements Management"* and "*Configuration Management*" processes.

2.1.1 Customer Requirements

Customer requirements are defined as "Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, environment, constraints, and measures of effectiveness and suitability (MOE/MOS)." [39].

CMMI defines **customer requirements** as "The result of eliciting, consolidating, and resolving conflicts among the needs, expectations, constraints, and interfaces of the product's relevant stakeholders in a way that is acceptable to the customer." [37].

The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing [39]:

- *Operational distribution or deployment*: Where will the system be used?
- *Mission profile or scenario*: How will the system accomplish its mission objective?
- *Performance and related parameters*: What are the critical system parameters to accomplish the mission?
- *Utilization environments*: How are the various system components to be used?
- *Effectiveness requirements*: How effective or efficient must the system be in performing its mission?

- *Operational life cycle*: How long will the system be in use by the user?
- *Environment*: What environments will the system be expected to operate in an effective manner?

2.1.2 Software Requirements Specification

CMMI defines **software requirements** as "A refinement of the customer requirements into the developers' language, making implicit requirements into explicit derived requirements." [37].

Software requirements specification (SRS) is a complete set of the behavior of the software including functional and non-functional (supplementary) requirements. Functional requirements are the specifications that describe functions, operations, constraints and give definitions of the software through use-cases, business rules, data definitions and other possible types specified according to the whole set of software requirements. Non-functional requirements are defined as "*requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints)*".

IEEE 830-1998 [40] proposes a standard approach for software requirements specification which describes possible structures, desirable contents, and qualities.

2.1.3 Business Rule

Business rule is defined as "*A* statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business." [39].

These rules are then used to help the organization to achieve goals better, communicate among principals and agents, communicate between the organization and interested third parties, demonstrate fulfillment of legal obligations, operate more efficiently, automate operations, perform analysis on current practices, etc. [39].

Ambler defines **business rule** as "A policy your software must satisfy. Business rules are what a functional requirement "knows," the controls and guidelines that are fulfilled by the functional requirement. An operating principle or policy of your organization." [36].

Gottesdiener told that “Business rules provide the knowledge behind any and every business structure or process. They are therefore at the core of functional requirements.” [21].

2.1.4 Use Case

Use case is defined as “A use case in software engineering and systems engineering is a description of a system’s behavior as it responds to a request that originates from outside of that system.” [39].

Ambler states that “A **use case** is a sequence of actions that provide a measurable value to an actor. Another way to look at it is that a use case describes a way in which a real-world actor interacts with the system.” [36].

2.1.5 Requirements Traceability

There are fourteen different types of traceability relations. Requirements traceability can be thought as traceability relations between requirements and any other kind of artifacts. These are [1]:

- Stakeholders – Requirements
- Stakeholders – Design
- Stakeholders – Code
- Stakeholders – Others (e.g. goal documentation, test cases, rationale and purpose documentation, etc)
- Requirements – Requirements [12], [13], [14], [16], [19], [20], [23], [24], [26], [28], [30], [31], [32], [35]
- Requirements – Design
- Requirements – Code
- Requirements – Other
- Design –Design
- Design – Code
- Design – Other
- Code – Code
- Code – Other
- Other – Other

Gotel and Finkelstein states that "**Requirements traceability** refers to the ability to describe and follow the life of a requirement, in both forwards and backwards." [7].

Pinheiro and Goguen states that "**Requirements traceability** refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the trace left by those elements on requirements." [28].

CMMI defines **requirements traceability** as "A discernable association between requirements and related requirements, implementations, and verifications." and bidirectional traceability as "An association among two or more logical entities that is discernable in either direction (i.e., to and from entity)." [37].

2.1.6 CMMI

There are mainly three processes of CMMI regarding requirements, traceability and impact analysis concepts. The following table [37] gives a brief description about these processes. Moreover, some specific practices of these processes are emphasized to make the connection between the terms used and CMMI clearer.

Table 1. Processes of CMMI

Process Name	Purpose of the Process	# Specific Practice	Description	Artifact
Requirements Development (RD)	Produce and analyze customer, product, and product component requirements	1.2	The various inputs from the stakeholders must be consolidated, missing information must be obtained, and conflicts must be resolved in documenting the recognized set of customer requirements. The customer requirements may include needs, expectations, and constraints with regard to verification and validation.	Customer requirements
		2.1	The customer requirements may be expressed in the customer's terms and may be nontechnical descriptions. The product requirements are the	Software requirements

Table 1 (continued)

Process Name	Purpose of the Process	# Specific Practice	Description	Artifact
			expression of these requirements in technical terms that can be used for design decisions.	
Requirements Management (RM)	Manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products	1.3	As needs change and as work proceeds, additional requirements are derived and changed may have to be made to the existing requirements. It is essential to manage these additions and changes efficiently and effectively. To effectively analyze the impact of the changes, it is necessary that the source of each requirement is known and the rationale for any change is documented.	
		1.4	The intent of this specific practice is to maintain the bidirectional traceability of requirements for each level of product decomposition. When the requirements are managed well, traceability can be established from the source requirement to its lower level requirement and from the lower level requirements back to their source. Such bidirectional traceability helps determine that all source requirements have been completely addressed and that all lower level requirements can be traced to a valid source.	

Table 1 (continued)

Process Name	Purpose of the Process	# Specific Practice	Description	Artifact
Configuration Management (CM)	Establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits	2.1	Change requests address not only new or changed requirements, but also failures and defects in the work products. Change requests are analyzed to determine the impact that the change will have on the work product, related work products, budget, and schedule.	

2.2 LITERATURE SEARCH

Ramesh and Edwards indicated necessary people, software artifacts and entities for developing a requirements traceability model in 1993 [10]. In this work, requirements were behaved as entities that can impact on each other. Hence, one of the first reasons for forming requirements traceability models appeared.

Gotel and Finkelstein made several important descriptions and an overview of requirements traceability problem in 1994 [7]. This work reports that, majority of the requirements traceability problems were due to inadequate pre-requirements specification traceability. They put requirements traceability into two groups as shown in Figure 1 taken from [29]:

- **Pre-RS traceability** refers to those aspects of a requirement's life prior to its inclusion in the requirement s specification.
- **Post-RS traceability** refers to those aspects of a requirement's life that result from inclusion in the requirements specification.

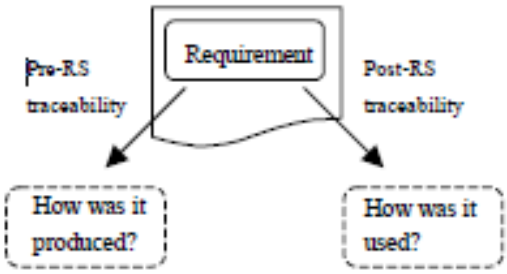


Figure 1. Pre-RS and Post-RS

Pre and post requirements specification distinction is not optimized for the reason that at software design, implementation and deployment time; developers find out software requirements on their own understanding of the problem. End-users can not address all possible needs. Hence, pre-requirements specification traceability must be sensitive to contextual needs. These are reasons why change requests and more reactive types of software requirements to change requests, business rules, have been evolved. As there are several requirements changes in today's software projects, this distinction creates several lifecycles for each requirement. Today, pre and post requirements specification can be re-expressed as *lazy* and *eager* generation of software requirements for the reasons told above.

- **Eager generation** is creation of software requirements before using them. These requirements may be changed during design and other phases.
- **Lazy generation** is creation of software requirements during use of other software requirements. These requirements are generated due to change requests or different prototypes.

Wieringa divided traceability into two groups in 1995 [34] as shown in Figure 2 taken from [29]:

- **Forward traceability** is the ability to trace a requirement to components of a design or implementation.
- **Backward traceability** is the ability to trace a requirement to its source, i.e. to a person, institution, law, argument, etc.

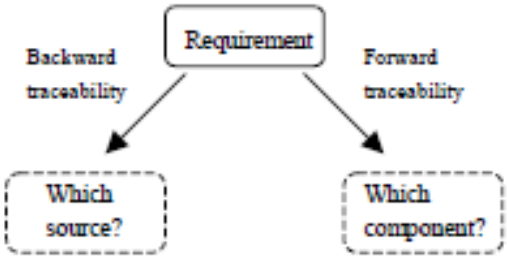


Figure 2. Backward and Forward Traceability

Lindval and Sandahl described traceability in two groups, again in 1996 [25] as shown in Figure 2 taken from [25]:

- **Vertical traceability** is tracing dependent items within a model.
- **Horizontal traceability** is tracing correspondent items between different models.

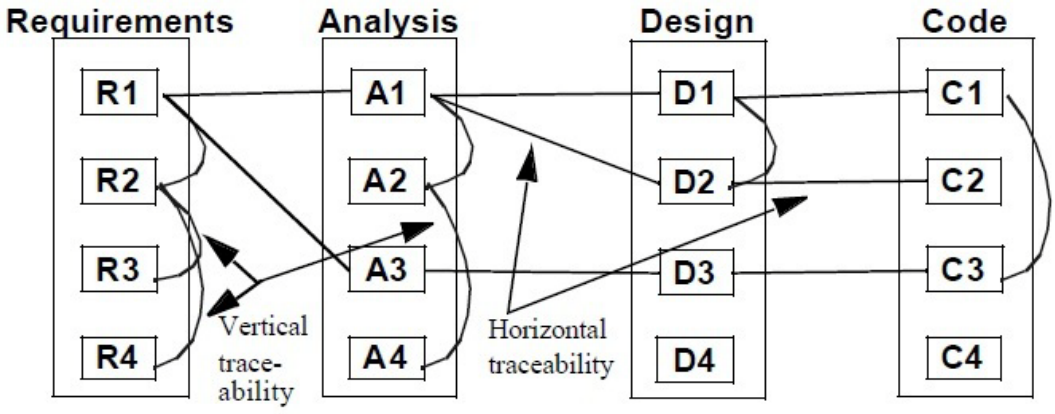


Figure 3. Vertical and Horizontal Traceability

Backward and forward traceability are similar to horizontal traceability. Most of the research on requirements traceability was concentrated on horizontal traceability as its granularity level is upper than vertical traceability.

After specifying the general descriptions and problems in the domain, research began to concentrate on creating requirements traceability models in 1995. Ramesh, Dwiggins and Edwards made a basic and semantic approach to requirements traceability models [5]. Proposed model described uses "*derived-from*" link for requirements. Two ends of this

relation can be described as customer requirements and software requirements in today's terms.

Research concerning details about requirements traceability problems and defining extensions is called **extended requirements traceability.** Haumer defines extended traceability as "*The relationship between recorded real world observations and parts of conceptual models.*" [11].

Gotel and Finkelstein extended the requirements traceability problem [6]. They stated that, literature focused on known requirements traceability problems and seeking more powerful traceability tools without trying to discover the problems at the core. They proposed setting up a shared, consistent and coherent requirements traceability scheme for each project, then commitment to the scheme from all the stakeholders, coupled with the need for some overall co-ordination.

Towards 2000's, researchers realized that requirements management and traceability concepts should be thought and evaluated together with the industry by empirical investigations. In 1998, Ramesh made a survey about the understanding of traceability by the people in the software industry [4]. He realized that the maturity of the organizations is parallel to the deployment degrees of traceability. He said that "*Managers of one organization that moved from Level 1 to Level 3 of the SEI CMM strongly believe their comprehensive traceability practice ("well beyond the narrow interpretation of CMM requirements") was an important factor in achieving this goal.*" Moreover, he called mature managers as "*high-end user managers*" and stated that

> *High-end user managers, in contrast, are committed to traceability as a mechanism for improving and maintaining the quality of the systems development process and see strategic benefits of incorporating traceability, even when it is not required by the project sponsors.*

Researchers also realized the importance of business rules towards 2000's. Business rules groups and web sites began to be formed. Gottesdiener told that, since business rules are behind functional requirements, without explicit guidance, software developers may not see business rules and make assumptions about conditions, policies and constraints. This can cause in unexpected business results. There should be standard taxonomy or categories for business rules [21].

In 2005, there was the same problem. Lindquist said that, analysts reported that reason for the failure of 71 percent of failed software projects was poor requirements management [9]. Lindquist also gave an example of real project which was about to fail. That project produced the application at one-quarter the cost and with fewer than 10 percent of the expected defects compared with outside development estimates by granular traceability. Approach was:

> *take a piece of code and quickly trace it back through the development process, back to requirements and then—rather than stopping there—map it all the way back to every affected business process to better gauge the application's impact on the business and to find hidden stakeholders.*

Reference traceability models are important as they address general problems and propose widely accepted solutions. They can be used as a basis for the construction of particular models. They save time and effort [29].

A general requirements reference model was proposed by Gunter et al. in 2000 [22].

Ramesh and Jarke proposed a requirements specific reference traceability model which has three basic elements in 2001 [32]:

- *Stakeholder*: People who have an interest on requirements.
- *Source*: The origins of a requirement and the artifacts.
- *Object*: Object being traced.

Mohan and Ramesh formed a knowledge management system [26] in 2002 based on the reference traceability model described in [32].

Business Rules Group declared a *Business Rules Manifesto* in 2003 describing some major aspects of business rules and related processes [41].

Cleland-Huang, Chang and Christensen proposed a new method of traceability called "*Event-Based Traceability"* based upon *event-notification* in 2003. This method is applicable in globally distributed development environments. Traceable artifacts are linked through an event service [8]. Notification method is used in current requirements management and traceability tools. This method is very helpful to developers responsible from different kinds of requirements artifacts as they can learn the changed artifacts without any guidance from the developers who is the source of the change. However, software requirements should be effectively separated in different kinds to get advantage of such a scheme.

Pinheiro divides traceability into two groups [29]:

- **Inter-requirements traceability** refers to the relationships between requirements. Inter-requirements traceability is important for requirements change and evaluation. It is used, for example, when extracting all requirements derived from a specific requirement or its chain for refinement.
- **Extra-requirements traceability** refers to the relationships between requirements and other artifacts.

Inter-requirements traceability is similar to vertical traceability. Its granularity level is lower than extra-requirements traceability. Today research concerning this kind of traceability is gaining attention.

Pinheiro states that a software traceability model should not be complex to be efficiently used, as forming traces are at least as important as traces themselves. There are three aspects that should be covered by a traceability model [29]:

- *Definition*: The *definition* is related to the specification of the traces and traceable objects.
- *Production*: The *production* is related to the capture of traces, usually by means of an explicit registration of the objects and their relationships.
- *Extraction*: The *extraction* is related to the actual process of tracing, i.e., the retrieval of registered traces.

Spanoudakis and Zisman tells that research into software traceability has been concerned with four areas [1]:

- The study and definition of different types of traceability relations.
- The provision of support for their generation.
- The development of architectures, tools and environments for the representation and maintenance of traceability relations.
- Empirical investigations of organizational practices regarding the establishment and deployment of traceability relations during the software development life cycle [4], [7], [15], [17], [25], [32], [33].

Traceability relations can be used for different purposes [1]:

- Change impact analysis (establish the impact that potential changes in some part of the system may have in other parts) and management (make decisions about whether or not such changes should be introduced, and with what priority) [18].
- System verification, validation, testing and standards compliance analysis.

- The reuse of software artifacts.
- Software artifacts understanding.

Filho proposed a new traceability technique that defines dependency links observed in the relationships among business rules in 2010 [27]. He described "*business-specific concerns*", related to stakeholders interested in checking if other business rules are impacted by the change; "*software-specific concerns*", those related in stakeholders interested in the impacted software artifacts. He aims to discover right requirements impacted from change of business rules.

2.2.1 Relating Evolving Business Rules To Software Design

Wan-Kadir and Loucopoulos proposed an approach that considers business rules as a part of a software system in 2003 [2]. They developed the "*Business Rule Model"* to specify business rules and "*Link Model"* to relate business rules to software design elements. They aimed to improve requirements traceability in software design called "*linking conceptual specifications of business rules to software designs*" with minimizing the cost of changes of business rules.

Their approach is called "*Manchester Business Rules Management (MBRM)*". The MBRM approach covers four software development stages which are centered on a business rules paradigm. These stages are; *elicitation*, *representation*, *mapping* and *implementation*. Work focuses on the mapping stage. They use "*business rule model*" to indicate business rules in a formal way. They use "*link model*" to map business rules to software design and implementation. They use abstractions in the implementation to separate business rules from other parts of the software and to decrease the impacts of business rule changes.

This approach is similar to a business rules engine structure primarily focusing on the business rule part of the software. This work tells they separated business rules; hence changes to business rules can be easily made in the design and implementation.

Entry point to software changes should be not only business rules, but also other software requirements as well. A traceability or relation model should cover all the software requirement types and form the relations more clearly.

2.2.2 Improving Software Quality Through Requirements Traceability Models

Salem aimed to develop an efficient and dynamic requirements traceability model to be used in small and large-scale projects in 2006 [3]. This model is composed of a "*Traceability Engine Component (TEC)*", a "*Traceability Viewer Component (TVC)*" and a "*Quality Assurance Interface (QAI)*". Their target is to enable requirement coverage by code through the proposed requirements traceability model.

TEC reads requirements from requirements database and builds a traceability matrix by analyzing the code. TVC views the information gathered by TEC as acting like a client. QAI addresses validation and verification of requirements. They use a requirements and traceability repository to determine the covered requirements by the code. Firstly developer matches the requirements and the associated code through TVC, then QAI reviews and looks for any matching errors between the requirements and the code and uses some flags to indicate the traceability conditions of the requirements.

This mechanism is a double-check mechanism to ensure right traceability between the requirements and the code. It mainly aims to cover all the requirements by the code.

This model requires commitment and support from a group called as Quality Assurance Group. QAI interface should be very smart to associate the requirements and the code by reviewing the code, automatically.

CHAPTER 3

WORK

3.1 STRUCTURE OF THE REQUIREMENTS

There was a need to build a general understanding of the software requirements at the beginning of the project. As this need primarily focuses on functional requirements, three types were decided to be used; *use cases*, *business rules* and *data definitions*. For these types of software requirements, specific definitions were made with the development team. Because, these definitions may and should be different among projects according to the contexts, restrictions and targets. Hence, every project should think about and create their own templates and roadmap to form the types and the definitions of the software requirements that is going to be used in the project as described in Appendix C. Then, templates and naming conventions were formed in order to provide right usage of these requirement types and definitions; moreover lead the development team.

Every software requirement belongs to every development team member of the project to provide developer independency. This issue is also important for CMMI processes [37]. Software requirements can be modified or deleted by the development team. However, *owner* approach is used. Software requirements have an owner who is the *author* of that requirement. When there is a need to modify that requirement, responsible of the task or change request is that person. This prevents the potential chaos of software requirements changes. But responsible person may change according to the workload of the people and priorities of other change requests and tasks.

A metadata repository was decided to be used in order to keep the data definitions. Requirements management tool is used as the metadata repository. Every developer can reach the latest version of the data definitions easily. Metadata can be always updated according to changing requirements in this common repository. It is not usable and maintainable to keep the metadata as files. Because, there may be several different

versions of the metadata if a common repository is not used and this can easily cause software verification and validation problems.

Several business rules which the customer did not mention can be caught by thinking on data definitions. Data definitions are valuable sources to understand the general structure of the software. For instance, if a data definition is automatically assigned by the software, several business rules can arise in minds regarding how that data definition is managed by the software. This also shows that there is an impact of data definition changes on business rules.

Business rules can be implemented in different layers of the software like database, data access object (DAO), service or client. It is important to make a clear distinction between business rules; hence it can be possible to give right decisions about what is going to be affected due to a change request. Every business rule should give information about how it lives in the software. These attributes are satisfied through different aspects of the business rule approach used in the project.

Use case approach is a widely accepted approach for collecting and describing functional requirements. This project also takes advantage of use case approach by modifying and doing some extensions. As business requirements form a basis for user requirements and business rules drive user requirements; there is an impact of business rule changes on use cases.

Requirement management tools do not provide semantic linking in a detailed manner. They are limited to some reference link types that are acceptable for every software project. Therefore every project should create and manage its own understanding for links. This project has two meanings for links between software requirements, traceability in other words:

- **Impact traceability**: A software requirement change can affect another software requirement. For instance, a data definition change can affect another data definition, business rule or a use case. Its direction is shown by *arrows* in the proposed traceability model. This type of traceability is emphasized in this work.

- **Association traceability**: A software requirement change can not affect another software requirement, but it can break the trace between each other. For instance, a business rule change can break the trace between a data definition and itself, but

it can not change that data definition. Its direction is opposite of impact traceability. Association traceability is impact traceability when it is looked from the other side. It is a reverse logical look to the entry point of the traceability.

3.1.1 Use Cases

Software requirements were use case based in the projects before the related one; but pure use case approach did not satisfy all of the needs from the viewpoint of the developers. There are two different types of use cases in this project:

- **Detailed use case**: The template given in p.62 of [36] is used for this type of use case as shown in Appendix B. This type of use case is used when there is valuable information about the flow of events to accomplish some goal.

- **Create, retrieve, update and delete (CRUD) use case**: Use cases for CRUD operations are almost the same with detailed use cases if flows are ignored. It was decided that there is no need to use detailed use cases for CRUD operations in order to prevent possible maintenance tasks in the future for these similar operations. CRUD use case template used is shown in Table 2 and Table 3.

Table 2. CRUD Use Case Template

Use Case Name	Update Project Information
Use Case Description	User updates project information.
Actors	YKK
Pre-conditions	IKIS-MEET-BR001 IKIS-MEET-BR002

Traces of this CRUD use case are:

Table 3. Traces of CRUD Use Case

Trace From	IKIS-PRJI-DD001 IKIS-PRJI-DD002 IKIS-PRJI-DD003 IKIS-PRJI-DD004 IKIS-MEET-BR001 IKIS-MEET-BR002 IKIS-PRJI-BR004 IKIS-PRJI-BR006 IKIS-PRJI-BR002 IKIS-PRJI-BR004
Trace To	

Pre-conditions exist in "*Trace From*" as they are business rules. Moreover, there is not "*Trace To*" information of use cases according to the proposed traceability model.

3.1.2 Data Definitions

Their naming convention is: **IKIS-*<module_abbreviation>*-DD*<nnn>***.

Data definitions are a special kind of software requirements that can be defined as form items in web-based projects. They can be GUI elements or not. They can be thought as a specified kind of business rules as they hold information, rules, attributes and constraints.

An example data definition with corresponding fields is shown below in Figure 4 with details in Table 4. Fields with red labels are mandatory fields, while others are optional fields.

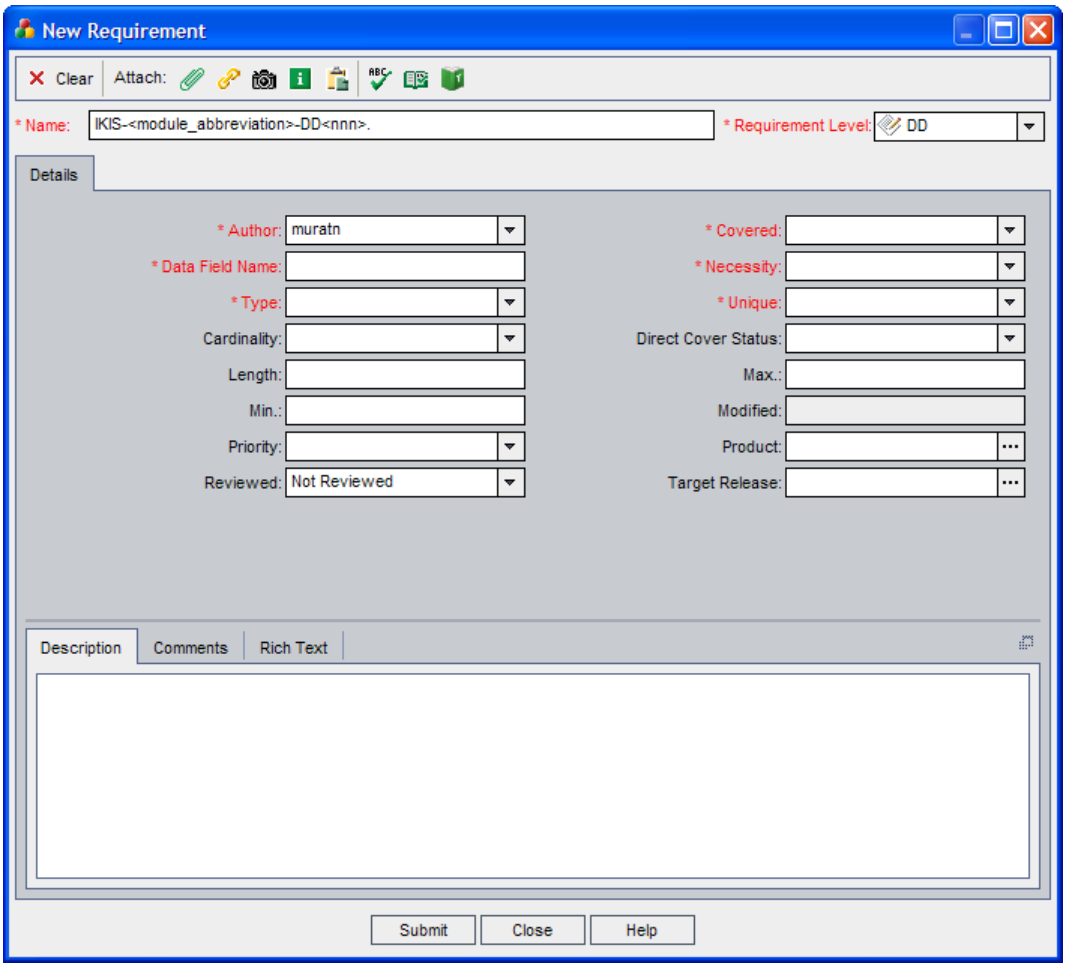


Figure 4. Data Definition Fields

Table 4. Data Definition Fields

Field Name	Mandatory / Optional	Description	Values
Name	Mandatory	The name with convention: IKIS-<module_abbreviation>-DD<nnn>.	
Author	Mandatory	The developer who created the data requirement.	
Data Field Name	Mandatory	Name of the data definition in the context of the software.	

Table 4 (continued)

Field Name	Mandatory / Optional	Description	Values
Type	Mandatory	Type of the data definition.	[Boolean, Date, Enumeration, String, Number, Time, Reference]
Necessity	Mandatory	Specifies whether this data definition is mandatory for the user. Rule Based data definitions become mandatory according to some business rules; hence they must have at least one business rule in their "Trace To". Approval Mandatory data definitions become mandatory when the user wants to approve the form.	[Mandatory, Approval Mandatory, Optional, Rule Based, N/A]
Unique	Mandatory	Specifies whether this data definition must be unique in the software.	[No, Yes]
Covered	Mandatory	Specifies whether this data definition is covered by qualification test procedures.	[No, Yes]
Length	Optional	Maximum length of a data definition that can be entered by the user for the ones of String type. Length can be used to hold precision and scale information of the data definition of Number type. For instance "*2,3"* tells that 2 digits can be entered before point and 3 digits can be entered after the point.	
Min	Optional	Minimum value of a data definition that can be entered by the user for the ones of Number type.	

Table 4 (continued)

Field Name	Mandatory / Optional	Description	Values
Max	Optional	Maximum value of a data definition that can be entered by the user for the ones of Number type.	
Cardinality	Optional	This field is meaningful in the context of the entity data definition belongs to. This field refers to other entities in general and used like foreign keys in databases.	[One, Many]
Priority	Optional	Priority of the requirement specified by the customer.	[Low, Medium, High]
Reviewed	Optional	Specifies whether the requirement is reviewed or not.	[Reviewed, Not Reviewed]
Direct Cover Status	Optional	Specified the last qualification test status of the requirement.	[Failed, N/A, No Run, Not Completed, Not Covered, Passed]
Target Release	Optional	Specifies the release that the requirement is going to be implemented.	[PROTOTİP-1, PROTOTİP-2, PROTOTİP-3, SON YAZILIM]

3.1.3 Business Rules

Their naming convention is: **IKIS-*<module_abbreviation>*-BR*<nnn>***.

Ambler proposes using "*BR#*" convention for identifying business rules uniquely. He tells that this unique identifier enables us to refer easily to business rules in other development artifacts, such as use cases [36].

Ambler tells that [36]

> *A rule of thumb is, if something defines a calculation or operating principle of your organization, then it is likely a good candidate to be documented as a business rule. You*

want to separate business rules out of your other requirements artifacts because they may be referred to within those artifacts several times.

Business rules are *rules* and *constraints* managed by the software which are specified by some data definitions impacting each other. Example business rules include the ones which do not have a direct relation with the user; trigger each other, perform some operations in the background like filtering, make items active or passive, add or remove some menu items, etc.

Business rules are not directly shown to the user like data definitions and they do not have a flow of events like use cases. For these reasons, there should be some references of business rules in the software which users can easily refer for the ones with user messages. These references are the names of the business rules shown in the messages with the convention: "BR*<module_abbreviation><nnn>".* Each business rule also has a user message in addition to this technical message or number that gives detailed information about the business rule.

An example business rule with corresponding fields is shown below in Figure 5 with details in Table 5 not shown in Table 4. Fields with red labels are mandatory fields, while others are optional fields.

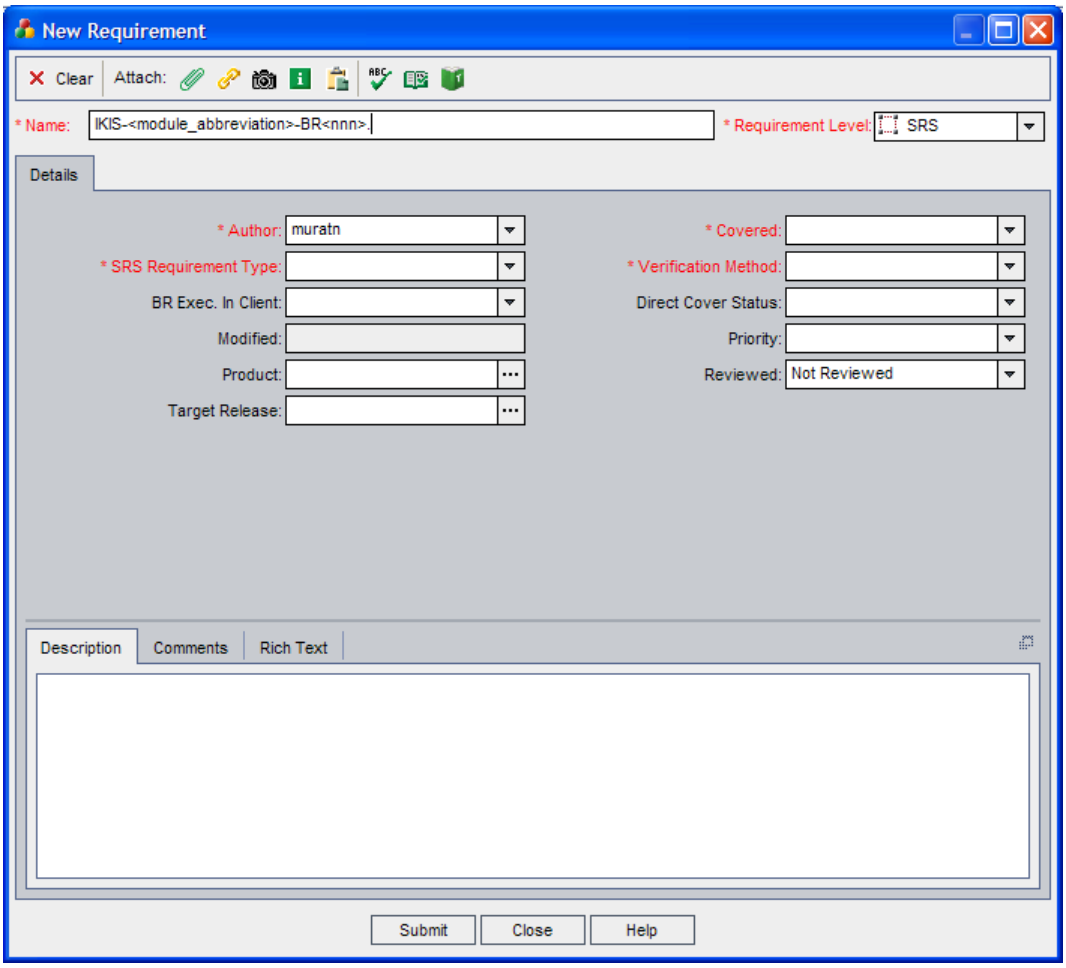


Figure 5. Business Rule Fields

Table 5. Business Rule Fields

Field Name	Mandatory / Optional	Description	Values
SRS Requirement Type	Mandatory	Software requirement type	Functionality
Verification Method	Mandatory	The verification type for the acceptance tests.	[Analiz, İnceleme (Inspection), İşlevsel Gösterim (Demonstration), Test]
BR Exec. In Client	Optional	Specifies whether this business rule will run on the client side or service side.	[N/A, No, Yes]

3.1.4 Validations

They are special kinds of business rules which have direct relation with the user through user messages. They validate user input, check user errors or warnings regarding authorization, constraints, etc.

The advised user message format is a sentence in passive form entered in the description field of the requirement in the form: "**Reason of error/warning + Result/Action taken**". If the user message does not give enough information to developers, all details should be clearly defined in the description field regarding developers.

Validations has two types: *Errors* and *Warnings*

- **ERRORS:**

  Their naming convention is: **IKIS-<*module_abbreviation*>-ER<*nnn*>**.

  Errors should be used in order to prevent the user doing some kind of false operations. There are generic errors regarding the properties of data definitions like *length*, *min*, *max*, *unique* and *necessity*.

- **WARNINGS:**

  Their naming convention is: **IKIS-<*module_abbreviation*>-WR<*nnn*>**.

  Warnings should be used to inform the user about an action or a situation which are not required to prevent from.

3.2 TRACEABILITY MODEL

Proposed requirements traceability model is shown in Figure 6. This traceability model can be thought as a "*inter-requirements traceability*" model [29]. According to this traceability model, there must be traces between a use case and the other types of requirements related to that use case like data definitions and business rules. Hence, there are data definitions and business rules in the "*Trace From*" of a use case and vice versa. Moreover, there must be traces between a business rule and the related data definitions to that business rule. Hence, there are data definitions in the "*Trace From*" of a business rule and vice versa.

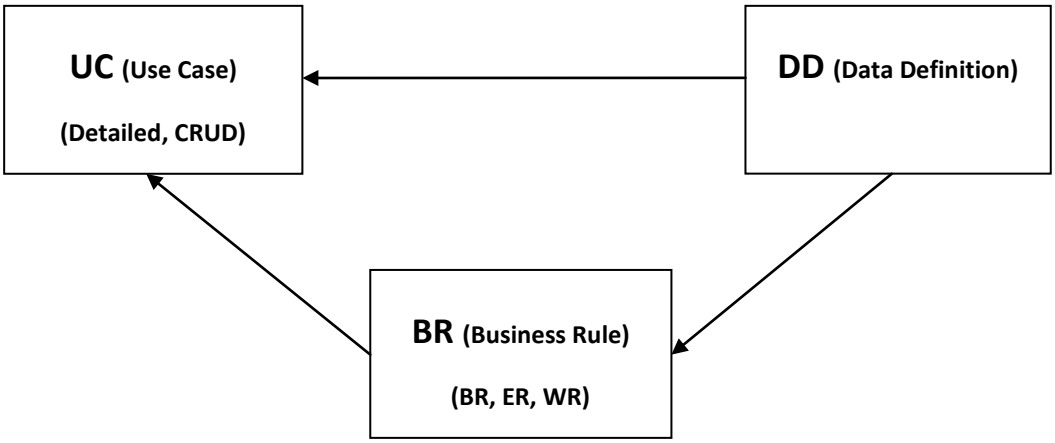


Figure 6. Proposed Traceability Model

Software requirements traceability provides two benefits called as *impact analysis*:

- Which software requirements change due to change requests?
- How users and software are affected due to change requests, hence user point of view to change requests.

3.3 CONSEQUENCES OF THE STRUCTURE AND THE TRACEABILITY MODEL

This part is about a few advantages of the proposed requirements traceability model throughout the related project. These items are not aimed to be evaluated formally as this thesis does not primarily focus on these items. However, *requirements driven design and implementation* item is evaluated in terms of workload efficiency gained in the evaluation part.

3.3.1 Requirements Driven Design And Implementation

Development team already had a software design prototype by having a requirements traceability model in software requirements. It is a good idea to form the design according to relations between different software requirements. If the dependencies between software requirements and the dependencies in the software design do not match, then *high cohesion* and *low coupling* [47] may suffer. Requirements traceability gives clues about the appropriate design. Therefore, if the development has a good software requirement model, they already have a good design. They should conform to each other; non-

conformance and tight vertical traces means bad design or indicates re-analysis of the requirements as shown in Figure 7.

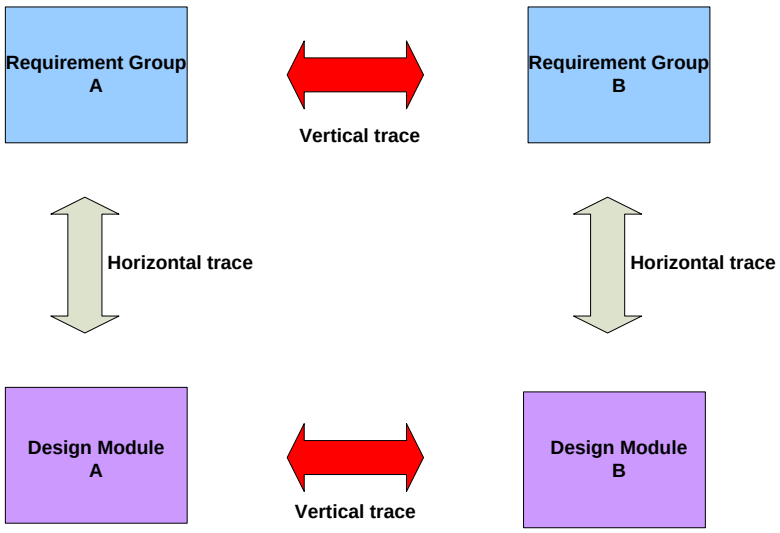


Figure 7. Relationship Between Requirements and Design

Software requirements are related to implementation in the following way:

- Data Definition → entities, database, test data
- Business Rule → business-rule engine, test data, qualification test procedures
- Use Case → service, graphical user interface (GUI), test data, qualification test procedures

It was tried to minimize the effect and the workload of change requests on implementation and tests when constructing the software design. All of the software requirements changes should not directly affect implementation at the same degree as software requirements. For instance, when a business rule code changes, the behavior of software should implicitly changes and this implicit behavior change is followed trough software requirements traceability. Therefore, impact analysis of change requests is made using software requirements before changing implementation.

If a change request comes to a business rule, the business rule should be the center of implementation change. It was thought that, not service side, but business-rule engine should know where and when it must work to provide independency. Hence, at business rule change requests, although traced use cases are impacted in software requirements,

only business rule code changes. We have a business-rule engine approach such that it can be even turned off without affecting anything in terms of code change. We thought such an approach by our software traceability model.

For the reason that many of change requests come to business rules and use case change is very costly; business rules should be isolated in the software requirements, design and implementation. This approach saved a significant amount of workload by affecting only the business-rule engine code; not the service code, graphical user interface code and the qualification test procedures.

3.3.2 Requirements Driven Architecture

Distribution of business rules and traces between software requirements give clues and make ease of *make-buy-reuse* analysis, architecture selection and high-level design. There may be more types; but three of them are listed below. These three types can live together:

- **Business Rules - User:** If there are several business rules interacting with the user, this is an indication for a *rich-client* application.

- **Business Rules – Data Definitions:** If there are several traces between business rules and data definitions, *transaction-management* and *database design* become important for properly working software.

- **Business Rules - Software:** If there are several business rules that the software is responsible of, then *business rule engine* may play an important role for the software. Hence, it is a good idea to look for effective ways of handling business rules.

3.3.3 Traceability Between Requirements And Code

One big challenge developers live is that when a change request comes, after doing necessary changes in the software requirements and design, it is hard to find the classes or code segments to be changed. Current requirement management tools do not give enough support to have such kind of information, indeed. It is much easier to make the necessary changes in the implementation by using the same names for business rules in the requirements and the implementation. Hence, business rules satisfy automatic traceability between software requirements and code by meaningful names.

3.3.4 Effective Tests

Every business rule is ready to become a test procedure. They are easily convertible to tests. Business rules play important role for the software. They should be carefully thought and tested. Writing test procedures and covering all the requirements through tests are very hard. Hence business rules are effective and easy way of writing tests and covering all of the requirements.

3.4 COMPARISON OF SIMILAR TRACEABILITY MODELS

A comparison was made in order to make a critic of the proposed traceability model. There are four different works including this work. The other works are; *Relating Evolving Business Rules to Software Design* [2], *Improving Software Quality Through Requirements Traceability Models* [3] and *Change Impact Analysis From Business Rules* [27]. The results are shown in Table 6.

- *R-D*: Requirements - Design
- *R-I*: Requirements - Implementation
- *R-R*: Requirements – Requirements

Table 6. Comparison of Similar Approaches

Work	Type	Formalized Business Rules	Business Rules Isolation and Change Independence	Requirements Driven Design	Requirements – Code Traceability	Auto Trace Creation
Wan-Kadir and Loucopoulos	R–D	Yes	N/A	Yes	Partial	No
Salem	R-I	No	N/A	No	Yes	Yes
Filho	R-R	N/A	Yes	No	No	No
This Work	R-R	Yes	Yes	Yes	Partial	No

CHAPTER 4

EXPERIMENTAL RESULTS AND EVALUATIONS

4.1 BACKGROUND

The proposed requirements traceability model was not relied on and evaluations of change requests were made on all types of software requirements of the regarding module in order to have a correct understanding on the advantages and disadvantages of the model.

Change requests in the configuration management tool and defects in the requirements management tool were primarily used to form the dataset. A snapshot of the requirements management tool is shown in Figure 8. The items used in the evaluation and descriptions of these items are listed below:

Change requests have attributes like;

- *Synopsis*: Short description of the change request.
- *Description*: Brief description, reasons and ideas about the change request.
- *Configuration item*: The source of the change request like *software requirements specification (SRS)*, *software design description (SDD)* or *Application Software*, etc. The configuration item is the entrance point of the change request's impacts to the whole software. For thesis purposes only change requests of which configuration item is "*SRS"* was used to form the dataset.
- *Links*: The items that changed while implementing the solution for the change request. These items can be *database change scripts*, *modified code*, *modified test data*, *modified test code*, *modified working reports*, *other relevant change requests*, *topics* and *other kinds of documentation*.

In order to hold the relevance between change requests and requirements, defects are used in the requirement management tool with related change request numbers. Defects have attributes like;

- *Requirement changing defect?:* A defect can affect a requirement or a qualification test procedure. If it affects a change request, then it is a *requirement changing defect*, otherwise it is a *qualification test procedure defect*.

- *Summary*: Short description of the defect. This information is the number of the relevant change request written with the convention "Change Request <*change_request_number*>".

- *Description*: Brief description, reasons and ideas about the defect. This information is empty for a requirement changing defect.

- *Severity*: The importance degree of the defect. If the defect is a requirement changing defect, this information is the same as the severity of the related change request.

- *Linked entities*: The items changed due to the defect. This information is bidirectional traceability between the changing item, requirement or test procedure, and the defect.

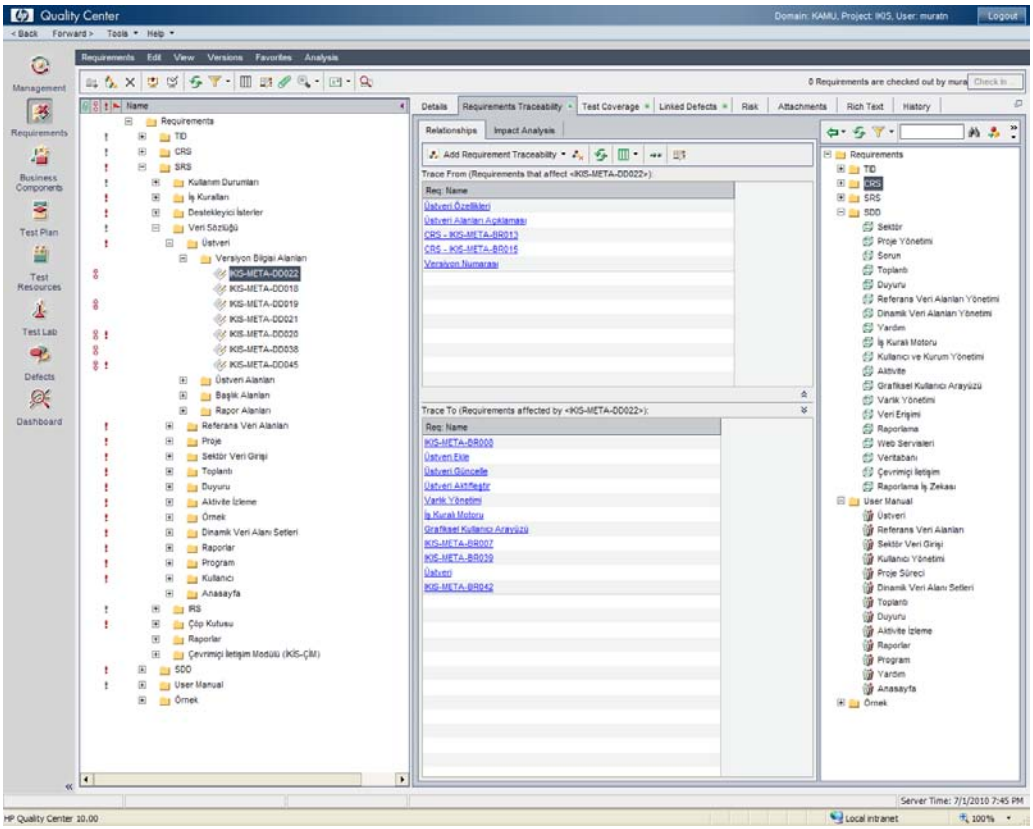


Figure 8. Requirements Management Tool

4.2 DATASET TEMPLATE

This is the dataset, shown in Table 7, used for the evaluation of the requirement traceability model and its results. Descriptions of the items are listed below:

Table 7. Dataset Template

Type of software requirement	What is impacted in the software requirements?			What is impacted in the implementation?			
DD, BR, UC	DD	BR	UC	Database	Class	Test data	Test class
	#	#	#	True / False	#	True / False	#
Aim: Change request distribution statistics	Aim: The correctness of requirements traceability model			Aim: The costs of implementation changes for each kind of change requests by finding average costs for each type and how much workload efficiency is gained.			

- *Type of software requirement*: Only change requests are used of which configuration item is "*SRS*" as the dataset was used to evaluate the requirement traceability model. The type of software requirements can be "*data definition (DD)"*, "*business rule (BR)"*, "*use case (UC)"* or any combination according to the requirement traceability model. This information is gained from *synopsis* and *description* attributes of the change request together. This information was used to determine change request distribution statistics.

- *What is impacted in the software requirements?*: The type of software requirements can be "*data definition (DD)"*, "*business rule (BR)"*, "*use case (UC)"* or any combination according to the requirement traceability model. This information is gained from requirements management tool by *linked entities* of the corresponding defect of the change request. Numbers of impacted software requirements are determined regarding the type of the requirement. This information is used to determine the correctness of the requirements traceability model.

  When there is a combination of different types, there appears a problem that it can not be strictly determined which software requirement type has affected which items. Assume that types of software requirements are "*BR, UC*" together and this change request has impacted on two business rule and one use case. This total effect is behaved as *true conformance* to the requirements traceability model. However, there is a possibility that, business rule impacted one business rule; use case impacted one use case and one business rule. This time, the first determination turns out to be false. This is a disadvantage of the evaluation.

- *What is impacted in the implementation?*: This information is gained from the links of the change request. Only database change scripts, modified codes, modified integration test data and modified test codes are taken into account. Database changes and test data changes are behaved as "*true*" or "*false*". Modified codes and integration test codes are counted as Java [46] classes. This information is used to find out the costs of change requests.

Preparation and maintenance of qualification test procedures are important and effort-resuming activities of a software project. But impacts of change requests on qualification test procedures were not taken into evaluation part as there is no data about the impacts of change requests on qualification test procedures. Qualification test procedures are maintained and corrected when there is a change on the related requirements. This is a deficiency of the evaluation.

4.3 ASSUMPTIONS

It is assumed that each database change and test-data change; each class change and test-class change require the same amount of effort in pairs while calculating the total workload efficiency of the proposed requirements traceability model.

4.4 EXPECTATIONS

It is expected that;

- (**1**) The number of change requests with software requirement type of *BR* is more than change requests with types of *DD* and *UC*.
- (**2**) Change requests with software requirement type of *DD* impacted *database* and *test-data* more than change requests with types of *BR* and *UC*.
- (**3**) The proposed traceability model is a correct way of making change request impact analysis by a major amount of efficiency of change request workload.

4.5 RESULTS AND EVALUATION

The dataset is given in Appendix A. Number of change requests according to the types of software requirements are:

- # Change requests: 403
- # Change requests of type DD: 97
- # Change requests of type BR: 230
- # Change requests of type UC: 35
- # Change requests of type DD, BR: 21
- # Change requests of type DD, UC: 3
- # Change requests of type BR, UC: 11
- # Change requests of type DD, BR, UC: 6

The amount of change requests of only one type (%) = 362 / 403 * 100 = 89,82 %. Hence the results of the evaluation have a reliability of ***89.82 %***.

Figure 9 shows the graphical distribution of change requests for the types of DD, BR and UC.

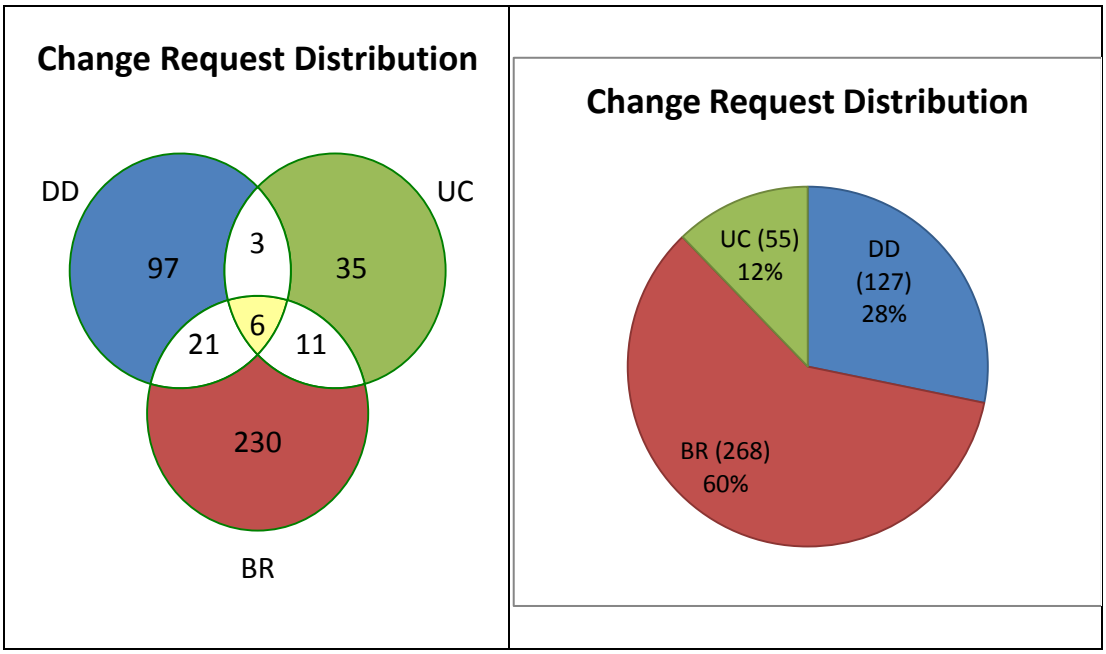


Figure 9. Change Request Distribution

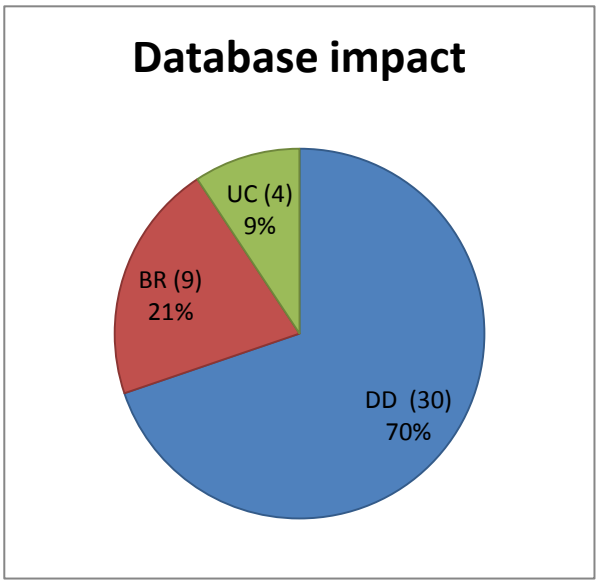
Table 8 shows the impacts of change requests on the database and the test data according to the types of software requirements. Figure 10 shows the graphical distribution for the types of DD, BR and UC.

Table 8. Database and Test Data Impacts

Type of change request	# Database impacts	# Test-data impacts	# Total impacts
DD	23	3	26
BR	2	8	10

Table 8 (continued)

Type of change request	# Database impacts	# Test-data impacts	# Total impacts
UC	0	1	1
DD, BR	3	1	4
DD, UC	0	0	0
BR, UC	0	1	1
DD, BR, UC	4	1	5



Changes of data definitions have a major impact on the database as there is a close relationship between the database and the data definitions.

Figure 10. Data Relevant Change Request Distribution

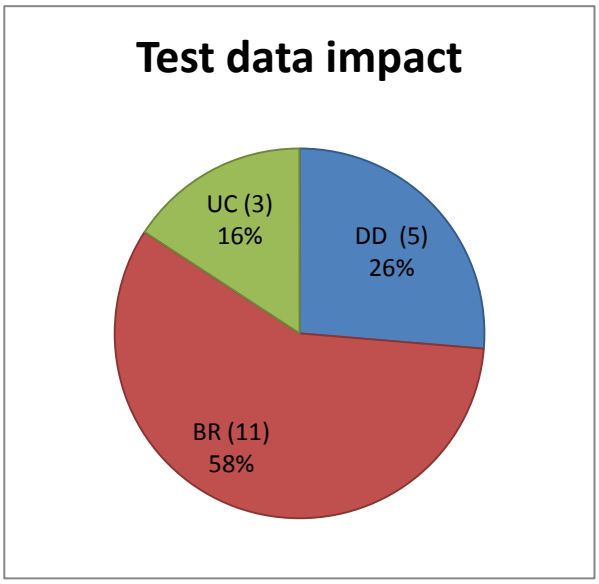
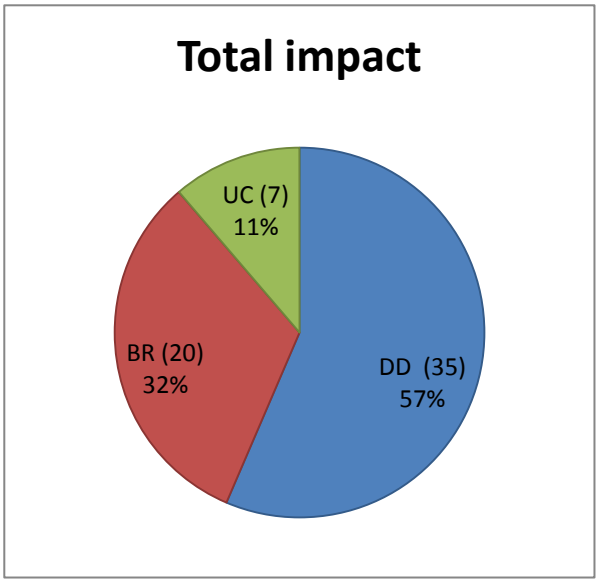
	Changes of business rule have a major impact on the test data as there is a close relationship between the tests and the business rules. This shows that business rules are of important elements of software testing.
	Use cases have a minimal impact on the database and the test data. Data definitions and business rules mainly impact different kinds of data relevant artifacts. Hence, it is a good idea to separate these three types of software requirements.

Figure 10 (continued)

The proposed requirements traceability model tells that;

- Change requests of type DD may affect requirements of type DD, BR and UC.
- Change requests of type BR may affect requirements of type BR and UC.
- Change requests of type UC may affect requirements of type UC.

Distribution of change requests according to conformance to the requirements traceability model are:

- # Change requests conforming to the proposed traceability model: 387
- # Change requests not conforming to the proposed traceability model: 16

$$\text{Correctness of the tracebility model (\%)} = \frac{\text{\# Change requests conforming to the traceability model}}{\text{\# Total change requests}} * 100$$

If the above formula is applied;

Correctness of the requirements traceability model (%) = (387 / 403) * 100 = ***96,02 %***

It was told that forming a software design according to the proposed traceability model facilitates making necessary changes in the implementation. Because, different types of change request affects only the same type of related code segment or module. Hence a significant amount of change request workload efficiency is gained. The related project uses a business-rule engine approach that is fully non-related to service implementation that gives necessary functions to the system defined in use-case steps. This business-rule engine structure has been built with the idea of most of change requests will be type of BR; therefore service implementation was affected less.

The workload efficiency gained with this approach can be calculated by looking at the "*What is affected in the implementation?*" part of the dataset. Table 9 shows the impacts of the change requests on the software implementation and test artifacts according to the types of software requirements. Table 10 shows the re-compiled version of Table 9 according to the types of DD, BR and UC.

Table 9. Impacts on Implementation and Test Artifacts

Type of change request	# Change requests	# Total database impacts	# Total class impacts	# Total test class impacts	# Total class, test class impacts
DD	97	26	301	10	311
BR	230	10	505	36	541

Table 9 (continued)

Type of change request	# Change requests	# Total database impacts	# Total class impacts	# Total test class impacts	# Total class, test class impacts
UC	35	1	139	3	142
DD, BR	21	4	62	3	65
DD, UC	3	0	22	0	22
BR, UC	11	1	90	2	92
DD, BR, UC	6	5	105	11	116

Table 10. Re-compilation of Impacts on Implementation and Test Artifacts

Type of Change Request	# Change Requests	# Total Database Effects	# Total Class, Test Class Effects	Average Database Effect	Average Class Effect
DD	127	35	514	0,275	4,047
BR	268	20	814	0,074	3,037
UC	55	7	372	0,127	6,763
TOTAL:		62	1.700		

By assuming that, every business rule change request would cost a use case change request in addition to the cost of itself, if we did not use the proposed traceability model; we can calculate the new average impact of a business rule by the following formula.

- Average database impact of a change request of type BR = 0,074 + 0,127 = **0,201**
- Average class impact of a change request of type BR = 3,037 + 6,763 = **9,800**

Table 11 was formed according to the new calculated average values.

Table 11. New Impacts on Implementation and Test Artifacts

Type of Change Request	# Change Requests	# Total Database Effects	# Total Class, Test Class Effects	Average Database Effect	Average Class Effect
DD	127	35	514	0,275	4,047
BR	268	**53,868**	**2.626,4**	**0,201**	**9,800**
UC	55	7	372	0,127	6,763
TOTAL:		**95,868**	**3.512,4**		

Despite of the fact that the new *#Total Database Effects* was calculated; it can not claimed that such a workload efficiency was gained as the database design was not formed regarding the proposed requirements traceability model.

We can calculate the change request workload efficiency as follows;

Change request workload efficiency gained (%) = (3.512,4 – 1.700) / 3.512,4 * 100= ***51,6 %.***

4.6 OTHER NOTES

The evaluation parts uses the change requests of which configuration item is "*SRS*" and expects that only this kind of change requests have impact on the software requirements. However, 50 out of 403 change requests are not of this kind. The developers who entered these change requests probably did not expect any impact on the requirements.

It can be said that, some requirements changes do not come directly from customers or developers; they come from implementation changes. This shows that not only requirements changes can impact implementation, but also implementation changes can impact requirements by an amount of ***12,4 %*** at the related project. Hence, bidirectional traceability between requirements and implementation becomes more important.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this thesis work, a software requirements traceability model approach was presented. The proposed model primarily tries to lead the software development teams make efficient and correct impact analysis on the change requests coming to software requirements from both customers and the development team. Moreover, the proposed model makes the development team save a significant amount of change request workload by the underlying software requirements structure which separates and isolates different types of software requirements focusing on the business rules.

Experimental results show that, most of the change requests came to the business rules with an amount of *60%* as expected. Moreover, data definitions were the software requirements which drove major amount of changes with an amount of *57%* on data relevant artifacts like the *database* and test *data*. These results show that, the proposed software requirements structure has an important role on identifying and isolating different kinds of software changes caused from change requests.

Analysis of the change request which came to software requirements and other software relevant change requests showed that, the proposed traceability model gives correct results with an amount of *96,02%*. There can be several reasons for the traceability model not to reach an amount of *100%*. Although the structure and the traceability model work well theoretically and practically; this model is used by the development team to form the software requirements and form the necessary traces between those. It is hard to make all the members of the development team to think exactly in the same way. In addition, forming software requirements requires intensive brainstorming and analysis. Hence, there is a possibility that the requirements were not identified correctly by each and every member of the development team. In other words, a property of a data definition that should be identified as a business rule might not be identified, or a step in a use case that

should become a business rule might not be specified or vice versa. There is a little amount of possibility that the requirements traces were not formed correctly. One more reason for this result can be the extension of change requests by the development team without specifying this fact in the change requests descriptions or evaluations. Therefore, the change requests might impact on more requirements or artifacts than they seem.

The proposed requirements traceability model was used to form the software design in order to make separation of modules consistently and isolate the infrastructure items in the software design that was thought to be points, on which the changes and impacts would focus. This requirements driven software design and usage of business rule engine provided change request workload efficiency with an amount of *51,6%*.

Preparation of the qualification test procedures are important and time-consuming activities. But, impacts of change request on test procedures could not be taken into account for the reason that there is not direct impact traces from change request to test procedures. The qualification test procedures are prepared according to the steps in use cases, mainly. But test procedures also examines business rules in an effective manner. At this point, one question arises in minds: "*Do business rule or use case changes have more impact on the test procedures?*". It seems that use cases should have. However, evaluation results show that, business rules are the major software requirements which impact test data used in integration tests with an amount of *58%.* By trying to answer this question, methodologies about the relevance between the software and the tests can be formed and used for effective planning of software test workload.

Applying a similar traceability model to the database design and artifacts may give good results like providing and maintaining the integrity of the database. Database design have items like *columns*, *primary keys*, *foreign keys*, *indexes*, *sequences*, *tables*, *views*, *procedures*, etc. All these items have close relationships with each other. Unfortunately, current database design tools do not have functionalities to notify the database designer about such concepts. By the time, database may lose its integrity or conventions specified before. Database relevant concepts should be analyzed and structures can be formed to detect inconsistencies in the database design while modifying the database. By this way, database design can take advantage of functions which requirement management tools have like traceability, notification, etc.

As mentioned before, customer is the core aspect of the software. The main source of all the software requirements is customer requirements. Hence, customer should also be an entity in the requirement traceability model. Traceability should go ahead to "*who said what*" while collecting customer requirements and analyzing them. An *owner* approach should be used for customer requirements. When a change request comes to a software requirement, that software requirement should be traced to the relevant customer requirement and the owner of that requirement. If the source of the change request and the owner does not match, extra effort should be spent to solve possible future customer dissatisfaction caused from changes without source customer approval.

In this thesis work, we presented a simple approach that can easily be applied to industrial software projects with possible current configuration and requirement management tools. Metadata repositories, business rules and requirement traceability are research areas which gain more attention, especially by the software industry as it benefits from these areas. Models like CMMI [37] are also driving forces for the organizations in order to apply configurable solutions for these concepts. The model proposed in this thesis work is a harmony of these concepts and models. As future works, some processes like *detection of traces*, *proposition of new software requirements* and *change request – defects integration* can be automated to decrease the workload and increase the effectiveness. In addition, the balance between *customer satisfaction* and these new developments should never be lost.

REFERENCES

[1] George Spanoudakis and Andrea Zisman, "Software Traceability: A Roadmap," *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, no. Recent Advancements, 2005.

[2] W. M. N. Wan-Kadir and Pericles Loucopoulos, "Relating evolving business rules to software design," *Journal of Systems Architecture*, vol. 50, no. 7, pp. 367-382, July 2004.

[3] Ahmed M. Salem, "Improving Software Quality Through Requirements Traceability Models," in *IEEE International Conference on Computer Systems and Applications*, 2006, pp. 1159-1162.

[4] Balasubramaniam Ramesh, "Factors Influencing Requirements Traceability Practice," *Communications of the ACM*, vol. 41, no. 12, pp. 37-44, December 1998.

[5] B. Ramesh, D. Dwiggins, G. DeVries, and M. Edwards, "Towards Requirements Traceability Models," in *Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems*, Tucson, AZ, USA, 1995, pp. 229-232.

[6] Orlena Gotel and Anthony Finkelstein, "Extended Requirements Traceability: A Framework for Changing Requirements," in *CAISE Workshop on Requirements Engineering in a Changing World*, Heraklion, Crete, Greece, 20-24 May 1996.

[7] Orlena C. Z. Gotel and Anthony C. W. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceedings of 1st International Conference on Requirements Engineering*, 1994, pp. 94-101.

[8] Jane Cleland-Huang and Carl K. Chang, "Event-Based Traceability for Managing Evolutionary Change," *IEEE Transactions On Software Engineering*, vol. 29, no. 9, pp. 796-810, September 2003.

[9] Christopher Lindquist. (2005, November) Fixing the Software Requirements Mess. [Online]. http://www.cio.com/article/print/14295

[10] Balasubramaniam Ramesh and Michael Edwards, "Issues in the Development of a Requirements Traceability Model," in *Proceedings of IEEE International Symposium on Requirements Engineering*, San Diego, CA , USA , 1993, pp. 256-259.

[11] P. Haumer, K. Pohl, K. Weidenhaupt, and M. Jarke, "Improving Reviews by Entended Traceability," in *Proceedings of 32nd Hawaii International Conference on System Sciences Volume 3*, Maui, Hawaii, 1999.

[12] A. Egyed, "A Scenario-Driven Approach to Trace Dependency Analysis," *IEEE Transactions on Software Engineering*, vol. 9, no. 2, pp. 116-132, February 2003.

[13] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105-127, 2004.

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

[15] P. Arkley, P. Mason, and S. Riddle, "Positon Paper: Enabling Traceability," in *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 61-65.

[16] J. Bayer and T. Widen, "Introducing Traceability to Product Lines," in *Proceedings of the Software Product Family Engineering (PFE): 4th International Workshop*, Bilbao, Spain, 2002.

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

[18] J. Cleland-Huang, Carl K. Chang, G. Sethi, K. Javvaji, H. Hu, J. Xia, "Automating Speculative Queries through Event-Based Requirements Traceability," in *Proceedings of the IEEE Joint International Requirements Engineering Conference*, Essen, Germany, 2002.

[19] P. Constantopoulos, M. Jarke, Y. Mylopoulos, and Y. Vassiliou, "The Software Information Base: A Server for Reuse," *VLDB Journal*, vol. 4, no. 1, pp. 1-43, 1995.

[20] O. Gotel and A. Finkelstein, "Contribution Structures," in *Proceedings of the 2nd International Symposium on Requirements Engineering (RE '95)*, 1995, pp. 100-107.

[21] Ellen Gottesdiener, "Capturing Business Rules," *Software Development Magazine: Management Forum*, vol. 7, no. 12, December 1999.

[22] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave, "A Reference Model for Requirements and Specification," *IEEE Software*, vol. 17, no. 3, pp. 37-43, May/June 2000.

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

[24] P. Letelier, "A Framework for Requirements Traceability in UML-based Projects," in *Proceedings of the 1st International Workshop on Traceability for Emerging Forms of Software Engineering (TEFSE '02)*, Edinburgh, UK, September 2002.

[25] M. Lindval and K. Sandahl, "Practical Implications of Traceability," *Software Practice and Experience*, vol. 26, no. 10, pp. 1161-1180, 1996.

[26] K. Mohan and B. Ramesh, "Managing Variability with Traceability in Product and Service Families," in *Proceedings of the 35th Hawaii International Conference on System Sciences*, Island of Hawaii, January 7-10, 2002.

[27] Antonio Oliveria Filho, "Change Impact Analysis From Business Rules," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, Cape Town, South Africa, 2010, pp. 353-354.

[28] F. Pinheiro and J. Goguen, "An Object-Oriented Tool for Tracing Requirements," *IEEE Software*, vol. 13, no. 2, pp. 52-64, March 1996.

[29] Francisco A. C. Pinheiro, "Requirements Traceability," in *Perspectives on software requirements*, Jorge Horacio Doorn, Ed.: Springer, 2003, ch. 5, pp. 91-113.

[30] K. Pohl, "PRO-ART: Enabling Requirements Pre-Traceability," in *Proceedings of the 2nd IEEE International Conference on Requirements Engineering (ICRE 1996)*, 15-18 April 1996.

[31] B. Ramesh and V. Dhar, "Supporting Systems Development Using Knowledge Captured During Requirements Engineering," *IEEE Transactions in Software Engineering*, vol. 18, no. 6, pp. 498-510, June 1992.

[32] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," *IEEE Transactions in Software Engineering*, vol. 27, no. 1, pp. 58-93, 2001.

[33] M. Strens and R. Sugden, "Change Analysis: A Step towards Meeting the Challenge of Changing Requirements," in *Proceedings of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, Fredrichshafen, Germany, March 1996, pp. 278-283.

[34] R.J. Wieringa, "An Introduction to Requirements Traceability," Faculty of Mathematics and Computer Science, University of Vrije, Amsterdam, September 1995.

[35] J. Dick, "Rich Traceability," in *Proceedings of the 1st International Workshop on Traceability for Emerging Forms of Software Engineering (TEFSE '02)*, Edinburgh, UK, September 2002.

[36] Scott W. Ambler, *The Object Primer*. Cambridge, UK: Cambridge University Press, SIGS Books, 2001.

[37] *CMMI for Development, Version 1.2*.: CarnegieMellon Software Engineering Institute, August 2006.

[38] (2010, Aug.) the Business Rules Group. [Online]. http://www.businessrulesgroup.org. [Accessed Aug 15, 2010].

[39] (2010, Aug.) Wikipedia. [Online]. http://www.wikipedia.org. [Accessed Aug 15, 2010].

[40] (2010, Aug.) IEEE Standards Description: 830-1998. [Online]. http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html. [Accessed Aug 15, 2010].

[41] (2003, November) Business Rules Manifesto. [Online]. http://www.businessrulesgroup.org/brmanifesto/BRManifesto.pdf. [Accessed Aug 15, 2010].

[42] (2010, Aug.) Business Rules Community. [Online]. http://www.brcommunity.com. [Accessed Aug 15, 2010].

[43] (2010, Aug.) Business Rules Oracle. [Online]. http://www.oracle.com/appserver/rules.html. [Accessed Aug 15, 2010].

[44] (2010, Aug.) Drools - JBoss Community. [Online]. http://jboss.org/drools. [Accessed Aug 15, 2010].

[45] (2010, Aug.) The BrBeans Framework. [Online]. http://publib.boulder.ibm.com/infocenter/wasinfo/v4r0/index.jsp?topic=/com.ibm.websphere.v4.doc/wasee_content/brb/concepts/cbrbfrmo.htm. [Accessed Aug 15, 2010].

[46] (2010, Aug.) Java Technology. [Online]. http://www.sun.com/java/. [Accessed Aug 15, 2010].

[47] Stevens W.P., Myers G.J. and Constantine, M., **"**Structured Design*," IBM Sytems Journal*, vol. 13, no. 2 pp. 115-139, 1974.

APPENDIX A

DATASET

The change requests of which "CR Number" is shown with red color are the ones not conforming to the proposed requirements traceability model. The change requests of which "Type of software requirement" is shown with blue color are the ones of which configuration item is not software requirements specification (SRS).

Table12. Dataset

No	CR Number	Type of software requirement	What is impacted in the software requirements?			What is impacted in the implementation?			
###	####	DD, BR, UC	DD	BR	UC	Database	Class	Test data	Test class
			#	#	#	True / False	#	True / False	#
1	130	UC	2	43	20				
2	131	DD	1	3	2				
3	132	DD, BR	1	0	1				
4	133	BR	0	0	0				
5	134	BR	2	0	0				
6	135	DD	1	0	0				
7	136	DD	15	1	5				
8	137	DD	1	0	0				
9	138	BR	0	1	0				
10	139	DD	3	0	0				
11	140	BR	0	1	0				
12	141	DD	132	2	0	TRUE			
13	142	DD	7	0	0				

Table 12 (continued)

14	143	DD	2	4	5				
15	144	DD	1	3	2				
16	145	BR	0	0	1				
17	147	DD	24	14	9				
18	148	DD	11	29	14				
19	149	DD, BR	22	5	40				
20	150	DD	6	3	2				
21	151	DD, BR	6	3	0				
22	153	DD	0	0	0	TRUE			
23	154	BR	1	0	0				
24	155	DD	1	0	0				
25	156	BR	0	0	1				
26	157	BR	0	1	0				
27	158	BR	0	1	0				
28	159	BR, UC	0	4	2				
29	160	BR	0	1	0				
30	161	BR	1	0	0				
31	162	BR	0	2	0				
32	163	BR	1	1	3				
33	165	DD	4	8	2	TRUE			
34	166	UC	0	8	0				
35	168	UC	0	0	1				
36	170	UC	0	0	1				
37	171	UC	0	0	2				
38	172	UC	0	0	1				
39	173	UC	0	0	4				
40	175	DD	1	0	0				
41	176	DD	6	2	4				
42	177	DD, BR, UC	4	12	4				
43	179	BR	0	9	7				
44	180	BR	0	1	0				
45	181	DD, BR	9	20	14				

Table 12 (continued)

46	182	UC	0	0	2				
47	183	DD	54	20	24				
48	184	DD	17	0	0	TRUE			
49	185	DD	140	0	0				
50	187	BR	0	0	60				
51	188	BR, UC	3	6	2				
52	190	UC	0	0	1				
53	192	DD	3	3	3				
54	193	DD	2	0	3				
55	194	DD, UC	2	0	1				
56	195	DD	1	0	0				
57	198	DD	9	0	0				
58	199	DD	2	1	0	TRUE			
59	200	DD, BR	19	0	4	TRUE			
60	201	BR	0	2	1				
61	203	BR	0	1	1				
62	204	BR	0	5	1				
63	205	BR	0	3	1				
64	206	BR	0	1	0				
65	207	BR	0	1	1				
66	208	DD	2	0	0	TRUE			
67	209	DD	1	0	0				
68	210	DD	0	1	1				
69	211	DD	2	0	0				
70	212	DD	1	2	2	TRUE			
71	213	BR	0	1	1				
72	214	BR	0	2	1				
73	216	DD, BR	1	7	0				
74	217	BR	0	4	2				
75	218	DD, BR	0	0	2				
76	219	BR	0	2	1				
77	220	BR	0	1	1				

Table 12 (continued)

78	221	BR	0	4	1				
79	222	DD, BR, UC	4	6	5	TRUE			
80	223	DD	1	0	0				
81	224	DD	16	0	2	TRUE			
82	225	BR	0	2	1				
83	227	BR	0	2	1				
84	228	DD	1	0	0				
85	229	BR	0	3	1				
86	230	DD, UC	1	1	2				
87	231	DD	0	0	10				
88	232	BR	0	0	0				
89	233	BR	0	2	1				
90	234	BR	0	2	1				
91	236	BR	0	1	1				
92	237	BR	0	4	2				
93	238	BR	0	5	2				
94	239	DD, BR	6	1	0				
95	240	DD, BR	1	1	0				
96	241	BR	0	1	0				
97	242	BR	0	1	1				
98	244	BR	0	4	0				
99	245	BR	0	2	0				
100	246	BR	0	2	0				
101	247	UC	0	0	4				
102	248	BR	0	2	0				
103	249	BR	0	5	0				
104	250	BR	0	2	0				
105	253	BR	0	4	2				
106	254	BR	0	1	0				
107	256	BR	0	4	0				
108	257	BR	0	3	0				
109	259	BR	0	1	0				

Table 12 (continued)

110	260	DD	2	0	0	TRUE			
111	261	BR	0	4	2				
112	263	BR	0	1	1				
113	264	BR	0	2	1				
114	266	BR	0	2	2				
115	267	BR	0	3	0				
116	270	BR	0	4	2				
117	271	BR	0	1	0				
118	273	BR	0	2	4				
119	274	BR	0	10	4				
120	275	DD, BR	1	2	2	TRUE			
121	276	BR	0	1	0				
122	277	BR	0	1	9				
123	280	BR	0	4	1				
124	281	BR	0	1	2				
125	282	BR	0	1	1				
126	283	BR	0	1	1				
127	285	DD, BR, UC	0	4	0				
128	287	BR	0	4	1				
129	289	BR	0	3	2				
130	290	BR	0	1	0				
131	291	BR	0	1	1				
132	292	BR	0	28	0				
133	293	BR	0	3	2				
134	294	BR	0	6	2				
135	295	BR	0	6	0				
136	296	UC	0	5	0				
137	302	BR	0	3	2				
138	303	BR	0	0	1				
139	304	DD	10	0	3	TRUE			
140	306	UC	0	1	1				
141	309	BR	0	1	0				

Table 12 (continued)

142	311	BR	0	3	2				
143	313	BR	0	1	1				
144	316	BR	0	1	1				
145	319	BR	0	4	1				
146	320	BR	0	5	2				
147	321	DD	2	2	0	TRUE			
148	322	BR	0	3	3				
149	326	DD	1	3	3				
150	327	BR	0	4	1				
151	329	BR	0	2	2				
152	330	DD	7	6	0	TRUE			
153	333	BR	0	1	0				
154	334	UC	0	2	2				
155	335	BR	0	2	3				
156	336	BR	0	1	3				
157	337	BR	0	2	1				
158	339	BR	0	2	0				
159	341	DD, BR, UC	0	7	14	TRUE			
160	344	DD	2	0	0				
161	348	BR	0	1	2				
162	350	UC	0	0	5				
163	354	DD	2	1	1				
164	360	BR	0	3	4				
165	362	BR	0	1	0				
166	363	BR	0	0	1				
167	370	BR	0	1	2				
168	**373**	**BR**	**0**	**4**	**3**		**8**		
169	376	UC	0	1	0				
170	377	BR	0	2	0		2		
171	378	BR	0	0	0				
172	380	BR	0	1	0		1		
173	381	DD	1	0	0				

Table 12 (continued)

174	385	DD	1	0	0				
175	386	BR, UC	0	1	0				
176	391	DD	1	0	0				
177	394	BR	0	3	0		1		
178	396	DD	1	2	0		5		
179	399	DD	1	0	0				
180	400	BR	0	2	0				
181	404	BR	0	2	0				
182	406	UC	0	0	2				
183	408	DD, BR	1	0	0				
184	410	DD	0	0	1				
185	417	DD	1	0	0				
186	418	DD	1	0	0				
187	419	DD	1	0	0				
188	420	DD	1	0	0				
189	428	DD	1	0	0	TRUE	3	TRUE	1
190	429	BR	0	2	2		4		
191	431	BR	0	1	1		3		
192	434	UC	0	0	1		3		
193	437	BR, UC	0	0	1		2		
194	440	DD	1	3	2				
195	442	UC	0	0	1				
196	443	BR	0	2	0		1	TRUE	1
197	444	BR	0	3	0				
198	445	BR	0	4	1		3		
199	446	BR	0	1	0		1		
200	448	DD, BR	1	2	2				
201	449	UC	0	0	1				
202	450	UC	0	0	0				
203	455	BR	0	1	0				
204	456	BR	0	0	0				
205	457	DD, BR	2	3	4		4		

Table 12 (continued)

206	462	BR	0	4	0				
207	465	BR	0	6	4		7		
208	472	BR	0	2	0		26		3
209	474	DD	4	0	0	TRUE	18	TRUE	
210	476	BR	0	2	0				
211	478	DD	3	0	0				
212	479	BR	0	0	0				
213	483	BR	0	1	1				
214	487	BR	0	5	6		4		
215	490	BR	0	1	0				
216	491	BR	0	1	0				
217	496	BR	0	1	0		1		
218	497	BR	2	1	1		2		
219	502	BR	0	0	1				
220	507	DD	1	0	0				
221	508	DD	1	0	0				
222	515	BR	0	0	1				
223	518	BR	0	1	0				
224	519	BR	0	6	0		2		
225	520	BR	0	1	0				
226	521	BR	0	1	0				
227	522	BR	0	0	0				
228	527	BR	0	1	1		3		
229	533	DD	0	0	0		6		
230	538	BR	0	1	0				
231	539	DD	2	0	0				
232	540	BR	0	4	4		2		
233	541	BR	0	1	0				
234	543	BR	0	1	0		2		
235	545	BR	0	1	0		3		
236	547	BR	0	1	0		1		
237	548	BR	0	1	0		1		

Table 12 (continued)

238	551	DD	0	0	0	TRUE	3		1
239	552	UC	0	1	0				
240	553	BR	0	1	0		1		
241	555	DD	0	0	0	TRUE	3		
242	557	BR	0	1	0				
243	558	BR	0	1	3		2		
244	559	BR	0	4	0				
245	560	BR	0	2	0		2		
246	563	BR	0	2	1		3		
247	564	DD	19	0	0				
248	568	BR	0	1	0				
249	570	DD	1	4	6		10		
250	573	DD	8	0	0		11		
251	575	BR	0	1	1		12		1
252	576	BR	0	2	0		4		
253	577	BR	0	1	1		3		
254	584	BR, UC	0	2	2				
255	586	DD	1	0	0				
256	587	BR	0	1	0		2		
257	588	BR	1	1	0	TRUE	3		
258	590	BR	0	2	0		3		
259	592	BR	0	6	0		1		
260	593	DD	3	0	0		2		
261	596	DD, BR	1	2	2		8		
262	597	BR	0	2	0		2		
263	598	BR	0	0	0				
264	602	BR	0	1	0		3		1
265	603	BR	0	2	0		2		
266	604	BR	0	1	2		2		
267	605	DD	1	0	0				
268	606	BR	0	1	1		4	TRUE	
269	609	BR	0	1	1		3		

Table 12 (continued)

270	610	DD, BR	2	2	4	TRUE	22	TRUE	2
271	612	BR, UC	0	2	19		15		
272	613	BR	0	1	0		1		
273	614	BR	0	0	0		2		
274	625	DD, BR	1	4	2		14		1
275	643	BR	0	3	3		4		
276	694	BR	0	3	1		4		
277	699	BR	0	2	7		5		
278	737	UC	0	1	2		1		
279	754	BR	0	3	2		1		
280	783	BR	0	1	0		1		
281	788	BR	0	5	0		5	TRUE	
282	795	BR	0	1	1		1		
283	797	DD, BR, UC	10	3	4	TRUE	61	TRUE	7
284	802	BR	0	1	0		3		
285	819	BR	0	1	0		1		
286	821	UC	0	0	1				
287	823	BR	0	2	1		8		2
288	826	UC	0	0	1				
289	829	BR	0	1	0				
290	838	BR	0	1	2		2		
291	844	UC	0	1	2		2		
292	850	BR	0	2	2				
293	853	BR	0	4	0		2		1
294	865	DD, BR	4	0	0				
295	878	DD	12	0	0		1		
296	879	DD	43	2	2	TRUE	57		
297	884	BR	0	1	0		2		
298	885	DD	1	0	0				
299	893	BR	0	2	0		2		
300	902	DD	9	0	0				
301	908	DD	10	1	1				

Table 12 (continued)

302	909	UC	0	0	0				
303	911	DD	8	0	0				
304	913	DD	9	0	0				
305	914	DD	1	0	0		1		
306	922	BR	0	1	0		1		
307	933	DD, UC	1	0	1		22		
308	937	DD	2	0	0	TRUE	1		
309	947	BR	0	2	0		3		
310	961	BR	0	1	3		1		
311	974	BR	0	1	0				
312	987	DD	2	0	0				
313	1002	BR, UC	0	2	1		9		
314	1008	BR, UC	0	1	1		3		
315	1009	DD	1	0	0		1		
316	1011	BR	0	3	0				
317	1039	DD	8	0	1				
318	1044	BR	0	1	1		1		
319	1059	BR	0	2	0		2		
320	1066	BR	0	20	0		32		
321	1080	BR	0	1	0				
322	1081	BR	0	2	2				
323	1084	BR	0	1	1				
324	1086	BR	0	1	0		3		
325	1099	BR	0	3	2		17	TRUE	4
326	1103	BR	0	1	1		12		
327	1109	BR	0	1	1		1		1
328	1113	DD, BR	2	4	0		5		
329	1114	DD	3	0	0	TRUE	32		
330	1115	BR, UC	0	0	2		12		
331	1116	BR	0	14	0		14		
332	1120	UC	0	0	11		12		
333	1126	BR	0	1	0		2		

Table 12 (continued)

334	1135	BR, UC	0	1	1		12		
335	1139	DD, BR	1	1	0		5		
336	1140	DD	3	0	0		4		
337	1142	UC	0	0	1		7		
338	1146	DD	2	3	0		3		
339	1147	BR	0	2	0		4	TRUE	4
340	1155	BR	0	3	0		6		
341	1163	DD	29	0	0		17		
342	1182	BR	0	1	0		1		
343	1196	BR	0	2	1		4		
344	1213	BR	0	1	1		33		
345	1214	BR	0	3	3		5		
346	1215	UC	0	8	0		11		
347	1216	DD	0	0	2				
348	1218	BR	0	1	0				
349	1227	BR	0	1	1				1
350	1228	DD	2	0	0		4		
351	1232	UC	0	2	0		1		
352	1240	BR	0	2	0		4		
353	1251	UC	0	0	2		30		
354	1253	UC	0	0	0		45		
355	1261	BR	0	2	0		2	TRUE	1
356	1276	BR	0	1	0		1		
357	1277	UC	0	1	2		27	TRUE	3
358	1296	BR	0	1	0		2		
359	1311	BR	0	21	0		29		1
360	1316	BR	0	1	0		4		
361	1317	BR, UC	0	1	0		37	TRUE	2
362	1322	BR	0	1	0		2		
363	1324	BR	0	3	1		4		2
364	1326	BR	0	3	1				
365	1327	BR	0	2	0	TRUE	19		

Table 12 (continued)

366	1336	DD	2	1	0		3		
367	1347	UC	0	0	1				
368	1357	BR	0	0	0		8		1
369	1364	DD, BR	1	0	1				
370	1365	BR	0	7	1		18		3
371	1367	DD	5	0	0		9		
372	1373	UC	0	0	1				
373	1378	BR	0	1	0		1		
374	1382	BR	0	1	0		1		
375	1388	DD	2	0	0		5		
376	1398	BR	0	1	1		7		
377	1403	DD	76	0	0	TRUE	19		
378	1432	DD	2	2	1	TRUE	20	TRUE	2
379	1444	BR	0	0	0		2	TRUE	1
380	1471	BR	0	4	1		6		1
381	1473	BR	0	1	0		1		
382	1474	DD	9	0	0				
383	1482	BR	0	4	0		4		
384	1500	BR	0	1	0		26		4
385	1501	BR	0	1	0				
386	1503	DD	1	0	0		3		
387	1510	BR	0	3	3		19	TRUE	3
388	1520	BR	0	1	0		1		
389	1526	BR	0	2	0				
390	1555	DD	3	0	5		34		6
391	1574	BR	0	9	0		12		
392	1582	DD	10	12	0				
393	1591	BR	0	1	0				
394	1602	BR	0	0	2		2		
395	1621	BR	0	1	0				
396	1642	DD, BR, UC	2	8	4	TRUE	44		4
397	1644	DD, BR	7	1	0		4		

Table 12 (continued)

398	1702	DD	8	0	0	TRUE			
399	1703	BR	0	6	0		11		
400	1716	BR	0	1	0				
401	1717	DD	2	0	0		17		
402	1718	BR	0	1	0		3		
403	1719	DD	6	0	0	TRUE	9		

APPENDIX B

USE CASE TEMPLATE

The following use case template was taken from Ambler [36].

Name: Enroll in Seminar

Description: Enroll an existing student in a seminar for which she is eligible.

Preconditions: The Student is registered at the University.

Postconditions: The student will be enrolled in the course she wants if she is eligible and room is available

Basic Course of Action:

1. A student wants to enroll in a seminar.
2. The student submits her name and student number to the registrar.
3. The registrar verifies the student is eligible to enroll in seminars at the university according to the business rule "BR129 Determine Eligibility to Enroll."
4. The student indicates, from the list of available seminars, the seminar in which she wants to enroll.
5. The registrar validates the student is eligible to enroll in the seminar according to the business rule "BR130 Determine Student Eligibility to Enroll in a Seminar."
6. The registrar validates the seminar fits into the existing schedule of the student, according to the business rule "BR143 Validate Student Seminar Schedule."
7. The registrar calculates the fees for the seminar, based on the fee published in the course catalog, applicable student fees, and applicable taxes. Apply business rules "BR180 Calculate Student Fees" and "BR45 Calculate Taxes for Seminar."
8. The registrar informs the student of the fees.
9. The registrar verifies the student still wants to enroll in the seminar.
10. The student indicates she wants to enroll in the seminar.
11. The registrar enrolls the student in the seminar.
12. The registrar adds the appropriate fees to the student's bill according to the business rule "BR100 Bill Student for Seminar."
13. The registrar provides the student with a confirmation that she is enrolled.
14. The use case ends.

Alternate Course A: The Student is Not Eligible to Enroll in Seminars.

A.3. The registrar determines the student is not eligible to enroll in seminars.

A.4. The registrar informs the student she is not eligible to enroll.
A.5. The use case ends.

Alternate Course B: The Student Does Not Have the Prerequisites.

B.5. The registrar determines the student is not eligible to enroll in the seminar she chose.
B.6. The registrar informs the student she does not have the prerequisites.
B.7. The registrar informs the student of the prerequisites she needs.
B.8. The use case continues at Step 4 in the Basic Course of Action.

Alternate Course C: The Student Decides Not to Enroll in an Available Seminar.

C.4. The Student views the list of seminars and does not see one in which she wants to enroll.
C.5. The use case ends.

APPENDIX C

PROJECT INFORMATION

The related software project used for the case study is *İKİS (İl Koordinasyon ve İzleme Sistemi). İKİS* is a two year, web based software project. The aim of *İKİS* is coordination and monitoring of investments and projects made by the government units; in addition taking information about city inventories.

İKİS is developed by *G222. G222* is a medium-sized, *CMMI Level 3* software development unit. It is one of the development units of *UEKAE (Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü, National Research Institute of Electronics and Cryptology). UEKAE* is an institute of *TÜBİTAK (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu, The Scientific and Technological Research Council of Turkey).* The customer is *DPT (Devlet Planlama Teşkilatı, State Planning Organization)*.