IDENTIFYING ARCHITECTURAL CONCERNS FROM NON-FUNCTIONAL REQUIREMENTS USING SUPPORT VECTOR MACHINE

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

BY

GÖKHAN GÖKYER

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

AUGUST 2008

Approval of the thesis:

IDENTIFYING ARCHITECTURAL CONCERNS FROM NON-FUNCTIONAL REQUIREMENTS USING SUPPORT VECTOR MACHINE

submitted by **GÖKHAN GÖKYER** 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. Volkan Atalay
Head of Department, **Computer Engineering** _____

Instr. Dr. Cevat Şener
Supervisor, **Computer Engineering Dept., METU** _____

Dr. Semih Çetin
Co-Supervisor, **Computer Engineering Dept., METU** _____

Examining Committee Members:

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

Instr. Dr. Cevat Şener
Computer Engineering Dept., METU _____

Instr. Dr. Meltem Turhan Yöndem
Computer Engineering Dept., METU _____

Instr. Dr. Ayşenur Birtürk
Computer Engineering Dept., METU _____

Yenal Göğebakan, M.Sc.
General Manager, CYBERSOFT _____

Date: 28.08.2008

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: Gökhan Gökyer

Signature :

ABSTRACT

IDENTIFYING ARCHITECTURAL CONCERNS FROM NON-FUNCTIONAL REQUIREMENTS USING SUPPORT VECTOR MACHINE

Gökyer, Gökhan

M.S., Department of Computer Engineering

Supervisor : Instr. Dr. Cevat Şener

Co-Supervisor : Dr. Semih Çetin

August 2008, 86 pages

There has been no commonsense on how to identify problem domain concerns in architectural modeling of software systems. Even, there is no commonly accepted method for modeling the Non-Functional Requirements (NFRs) effectively associated with the architectural aspects in the solution domain. This thesis introduces the use of a Machine Learning (ML) method based on Support Vector Machines to relate NFRs to classified "architectural concerns" in an automated way. This method uses Natural Language Processing techniques to fragment the plain NFR texts under the supervision of domain experts. The contribution of this approach lies in continuously applying ML techniques against previously discovered “NFR - architectural concerns” associations to improve the intelligence of repositories for requirements engineering. The study illustrates a charted roadmap and demonstrates the automated requirements engineering toolset for this roadmap. It also validates the approach and effectiveness of the toolset on the snapshot of a real-life project.

Keywords: Architectural Concerns, Machine Learning, Natural Language Processing, Non-Functional Requirements, Requirements Engineering, Support Vector Machine.

ÖZ

DESTEK VEKTÖR MAKİNESİ KULLANARAK İŞLEVSEL OLMAYAN GEREKSİNİMLERDEN MİMARİ İLGİLERİ TESPİT ETMEK

Gökyer, Gökhan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Instr. Dr. Cevat Şener
Ortak Tez Yöneticisi : Dr. Semih Çetin

Ağustos 2008, 86 sayfa

Yazılım sistemlerinde mimari modelleme için "Problem Kümesi İlgileri"nin nasıl ifade edileceği hakkında kesin bir yaklaşım mevcut değildir. Bunun yanı sıra "Çözüm Kümesi" içindeki mimari ilgi alanlarına yönelik olan "İşlevsel Olmayan Gereksinimleri" modellemek için de kabul edilmiş bir yöntem yoktur. Bu tez çalışması; "Destek Vektör Makinesi" tabanlı "Makine Öğrenimi" yöntemini kullanarak, işlevsel olmayan gereksinimleri mimari ilgi alanlarıyla ilişkilendirme sürecini otomatikleştiren bir yöntem önermektedir. Bu yöntem; İşlevsel Olmayan Gereksinim içeren yalın metinleri ayıklamak için alan uzmanlarının denetiminde "Doğal Dil İşleme" tekniklerini kullanır. Önerilen yaklaşımın değer katan yönü; bu süreçte Makine Öğrenimi tekniklerinin birbiri ardına kullanılmasıdır. Bu yaklaşım sayesinde başka problem kümeleri için önceden eğitilmiş olan bilgi depoları; gereksinim mühendisliği bünyesinde yüksek başarı seviyesine ulaşan "İşlevsel Olmayan Gereksinim - Mimari İlgi Alanı" eşleşmelerine dönüştürülmektedir. Tez çalışması; sistematik bir yol haritası ve bu yol haritası üzerine destekleyici bir "Gereksinim Mühendisliği Araç Seti" sunmaktadır. Ayrıca, tez çalışmasında önerilen yaklaşım ve geliştirilen araç setinin başarısını ortaya koymak için gerçek bir proje üzerinde elde edinilen deneyimler de sunulmaktadır.

Anahtar Kelimeler: Destek Vektör Makinesi, Doğal Dil İşleme, Gereksinim Mühendisliği, İşlevsel Olmayan Gereksinimler, Makine Öğrenimi, Mimari İlgiler.

To My Family

ACKNOWLEDGMENTS

Firstly, I would like to thank to my thesis supervisor Dr. Cevat Şener and co-supervisor Dr. Semih Çetin for their guidance, support and motivation they provided throughout my research. I also like to thank to Dr. Meltem Turhan Yöndem and Umut Eroğul for their guidance about the Natural Language Processing and Machine Learning parts in my study. I would like to thank to my mom, my dad and my sister for everything. Thanks to my girlfriend for her understanding me during my thesis study.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

AA	Architectural Aspect
AC	Architectural Concern
ACS	Architectural Concern Spaces
ADL	Architecture Description Language
AORE	Aspect-Oriented Requirements Engineering
AT	Architectural Tier
AV	Architectural View
CLIR	Cross Language Information Retrieval
FR	Functional Requirement
LEL	Language Extended Lexicon
ML	Machine Learning
NFR	Non-Functional Requirement
NFR2AC	Non-Functional Requirements to Architectural Concerns
NLP	Natural Language Processing
POS	Part-of-Speech
QA	Quality Attribute
SVM	Support Vector Machine
UCS	Utility Concern Spaces

CHAPTER 1

INTRODUCTION

1.1 Statement of the problem

Non-Functional Requirements (NFRs) are the needs of a system to carry out its operations under a set of constraints and quality expectations. That is why they are more oriented to “how” and “why” questions rather than “what”. Furthermore, capturing the NFRs and then matching them with architectural concerns are harder than dealing with functional requirements since users feel more difficulty in explaining the NFRs definitely at every stage of requirements modeling. All these put off system designers using more methodical techniques for managing the NFRs in contradiction to the case that they can use systematic approaches and tools to model the behavioral aspects of an information system [1].

Difficulties of modeling NFRs motivate researchers to explore new ways and techniques. Natural Language Processing (NLP), Machine Learning (ML), Aspect-Oriented Requirements Engineering (AORE), Speech Processing, Domain Specific Modeling, and Feature-Oriented Modeling are some of the methods to identify and model NFRs. These techniques can be occasionally harmonized, too. For example, Baniassad et al. [2] used aspects for modeling the features in terms of base and crosscutting themes, where NFRs play a significant role. Similarly, Loughran et al. [3] represented the feature models with aspects for product lines. Others such as proposed by Rosenhainer [4], combined the NLP with uses case based on domain modeling for mining the NFRs.

However, none of the previous studies achieved to represent NFRs in a system architecture model with an automatic transformation from the pure requirements. Some studies created their own frameworks for NFR representation. Nevertheless, such studies left as the set of examples for only using the framework in an abstract

level (See Chung [24]). Some of the others tried to add NFRs to the UML diagrams with new attribute insertions (See [9, 25]). But, almost all of these attempts lead to manual transformation under an expert control without proper level of automation.

1.2 Objective of the study

This thesis proposes an approach for identifying the NFRs from plain text by Natural Language Processing (NLP) techniques and mapping the identified NFRs to "architectural aspects" and "quality attributes" in the problem domain by Machine Learning (ML) techniques. Essentially, this work complements the architectural modeling approach with symmetric alignment of multiple concern spaces given in Cetin et al. [5], which proposes an inclusive architectural modeling method once the NFRs have been identified. This work fills the gap in that sense to guide the NFRs identification and automatic construction of the Utility Concern Spaces by mapping such NFRs to architectural aspects and quality attributes, respectively. Putting the "architectural aspects" and "quality attributes" together is referred to as "architectural concerns" throughout the thesis.

The contribution of this approach lies in continually applying the cycles of NLP techniques as part-of-speech tagging, phrase chunking, stemming and ML techniques as SVM against "NFRs – Architectural Concerns" associations collected from several business domains. This way of keeping smarter repositories continually guides the architect for making decisions effectively. Moreover, this approach provides automated tools for mapping the NFRs to architectural concerns as well. After briefly giving the related work, proposed approach will be explained and demonstrated on a real life case problem from the automotive domain.

1.3 Organization of the thesis

The next chapter presents the previous and related work, current standards and technologies. In Chapter 3, the proposed approach "Non-Functional Requirements to Architectural Concerns" (NFR2AC) is presented by explaining the roadmap and accompanied toolset in detail Continually, Chapter 4 defines the implementation

details and experimentation details where NFR2AC approach and the developed toolset have been applied on a case problem from a real-life application in automative domain. Eventually, Chapter 5 summarizes and concludes the thesis.

CHAPTER 2

BACKGROUND

In this chapter, the background information is given for the topics relevant to thesis study. The major related topics are listed as *Requirements Engineering*, *Non-Functional Requirements and Architectural Modeling*, *Natural Language Processing (NLP)*, *Machine Learning (ML), and Support Vector Machine (SVM).*

2.1 Requirements Engineering

Liu et al. [6] defines the *requirement engineering* as analyzing and understanding the problem domain effectively, in order to expose all the necessary information about the client requests, before developers start to implement the software system. It is also emphasized that the requirements engineering only targets the problem domain instead of design and implementation details.

2.1.1 Requirements

Robertson et al. [7] identifies the *requirements* as a set of responsibilities and qualities that a product must do or have. A requirement may occur in cases when it has to meet some functionality or some quality in the requested product, or when a client wants it as a part of that product. Therefore, all requirements must be specified before designing a product. Otherwise, the product cannot meet all the expectations defined by the domain, communities and actual users.

Requirements can be classified into two types: namely “Functional Requirements” and “Non-Functional Requirements”.

2.1.2 Functional Requirements

Functional requirements can be defined as the actions and functionalities that are supposed to be provided by the product, according to Robertson et al. [7]. They are

the functionality specifications and derived from the fundamental goal of the product, which satisfy the actions like check, calculate, record, retrieve etc. An example for the functional requirement is:

"The product shall warn if the scheduling date is neither today nor within the next two days."

This example informs us about some special case that is the scheduling date can never be more than two days in advance. For the derivation process of these requirements, product use cases are employed. Therefore, a scenario is written from the pieces of use cases and related functional requirements are derived from this scenario. In order to find new functional requirements, new use cases should be drawn. Moreover, functional requirements are business requirements and can be validated by the business people. As stated by Robertson et al. [7], if business people tell all the requirements for each use case, then enough functionality will be collected for a running product.

2.1.3 Non-Functional Requirements

Robertson et al. [7] defines the "Non-Functional Requirements (NFRs)" as set of properties and qualities that a product must contain in order to run fast, reliable, usable, attractive, that means successfully in short. Extracting more NFRs and modeling them improves the success and quality of a product. An example NFR can be:

"The product shell determine 'friend or foe' in less than 0.25 seconds."

Functional requirements are the fundamental requirements, which provide product running. However, NFRs are not required for basic execution but they suggest such fundamental requirements perform in a certain manner. They anticipate minimum and maximum conditions for these activities for issues like security, performance etc. They make these activities running in desired conditions and limitations according to Robertson et al. [7].

For example, video cameras should record all entrances and exits of the building in a bank security system. This states a functional requirement. However, if someone says that video camera system must send instantaneous video stream data within at most two seconds, then this requirement becomes a non-functional requirement.

2.2 Non-Functional Requirements and Architectural Modeling

Liu et al. [6] says that two vital activities in software life cycle are *requirements engineering* and *software architecting*. The requirements engineering aims at investigation and elicitation of the correct specifications of a system before developers start implementing. Contrastly, the aim of software architecting is constructing the architecture of the system due to the shape of the solution space in order to improve the success rate in development facilities.

Liu et al. [6] also states that there is a gap between mapping requirements to software architecture. Moreover, the mapping relationship is indirect and not straightforward in traditional software development strategies; therefore existing mapping strategies are insufficient for this mapping process.

Similarly, Franch et al. [8] identifies that software systems are combined from both functional and non-functional elements. However, most of the work has been done for the functional requirements. In fact, non-functional requirements are important as much as the functional ones. Cysneiros et al. [9] also claims that although both are primarily important, NFRs are more expensive and more difficult to deal with from the beginning of the software development process, and throughout the whole life cycle.

There are many software projects that fail right after deployment because of paying less attention to the non-functional requirements. Therefore, this study focuses on the non-functional requirements and tries to incorporate NFRs into components and connectors for architectural modeling. In particular, NFR issues such as security, system performance, data integration, data confidentiality and privacy might result

in complications and problems at the later stages of software projects, as Yusop et al. [10] indicates.

Franch et al. [8] defines the components and connectors as follows: *components* are dynamic computational units of a software system. Software modules (classes), libraries of reusable components, whole systems, clusters of similar system are all can be counted as examples of software components. Moreover, *connectors* can be defined as the units that provide the interaction and bridging process between components. Method invocation, network connection and data sharing are some of the examples of connectors. The connection points between components and connectors are identified as *ports*.

Relating the architectural model with requirements (especially NFRs) and deriving the architectural modeling are not straightforward. Moreover, the architectural model can be represented with more than one standard ways. Use of Architecture Description Languages (ADLs) and diagrammatic notations vary in the literature. An architectural model can be represented by using the lexicon first introduced by Kazman et al. [11] within the context of *Software Architecture Analysis Method (SAAM)*. Figure 2.1 depicts security framework architecture framework with a lexicon defined by SAAM and reveals merely the structural representation of an example architecture formed by the components and the connectors.

2.3 Natural Language Processing

Chowdhury [12] defines the "Natural Language Processing (NLP)" as a research and application area, which investigates the computer usage in understanding and manipulating natural language text or speech to obtain some benefits. Moreover, it is stated that NLP researchers try to find some samples on how human beings use language and then convert these samples into more proper representations for ease of computer understanding.

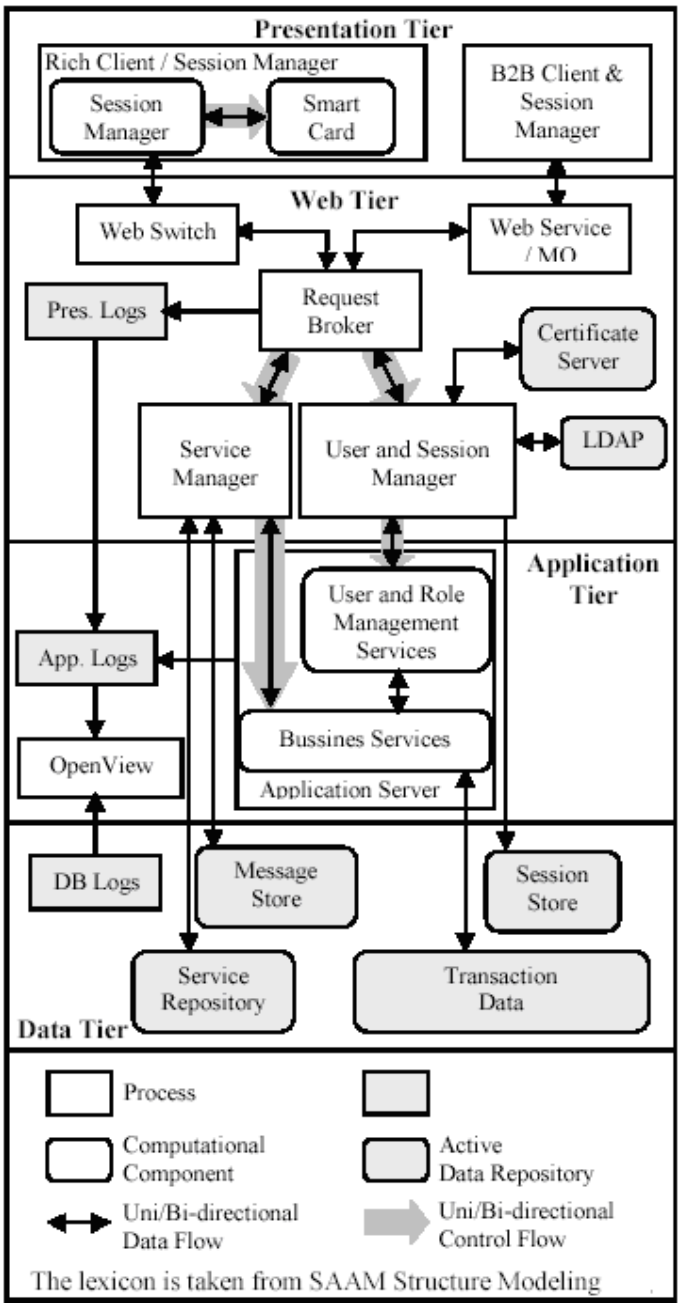

Figure 2.1 - An example of architectural model

Some fields of studies for NLP applications are sampled as machine translation, natural language text processing and summarization, user interfaces, multilingual and Cross Language Information Retrieval (CLIR), speech recognition, artificial intelligence and expert systems, and so forth. These application areas reveal the case that NLP is one of the best candidates for requirements engineering process, especially for capturing the NFRs.

In the related work of Chowdhury [12], it also claimed that every NLP task is built according to the natural language understanding issue. Here, there are three main problems for understanding the natural language in computer programs. The first problem is about the thought process, the second one is the representation and meaning of the linguistic input, and the third one relates to the world knowledge. Therefore, from the beginning to the end every NLP system should start studying at the word, then move on to sentence level, and finally conclude to the context level on the whole domain level. The reason is, first the morphological structure, nature (such as part-of-speech, meaning) etc. of the word has to be determined and after that the word order, grammar, meaning of the entire sentence and in conclusion domain knowledge, etc. should be resolute. There can be a specific meaning or connotation for each word and/or sentence in a given context or domain, and may have relations with many other words and/or sentences in the given context [12].

There are some methods used in NLP applications like Text Segmentation, Part of Speech Tagging, Stemming, Lemmatization, Phrase Chunking, Syntactic Parsing, Named Entity Recognition, Term Extraction, Text Summarization, Language Identification, Statistical Language Modeling, Corpora etc. All these applications specified in [12], are not used in this study, so only the used ones will be explained.

Part-of-speech (POS) tagging is defined in LingPipe [13] as the process of parsing each word as a token in each sentence and labeling each token with the most probable tag label, such as "adjective", "noun", “verb” etc. An example for POS tagging process is given in Figure 2.2. In this example, output of the POS Tagger is

"The/DD students/NNS had/VVB the/DD exam/NN in/II the/DD classroom/NN ./." the sentence of "The students had the exam in the classroom.". In this thesis, LingPipe is used for the POS Tagging module and inside the LingPipe, MedPost POS Parser [14] is used and the related tag set is given in Appendix A.

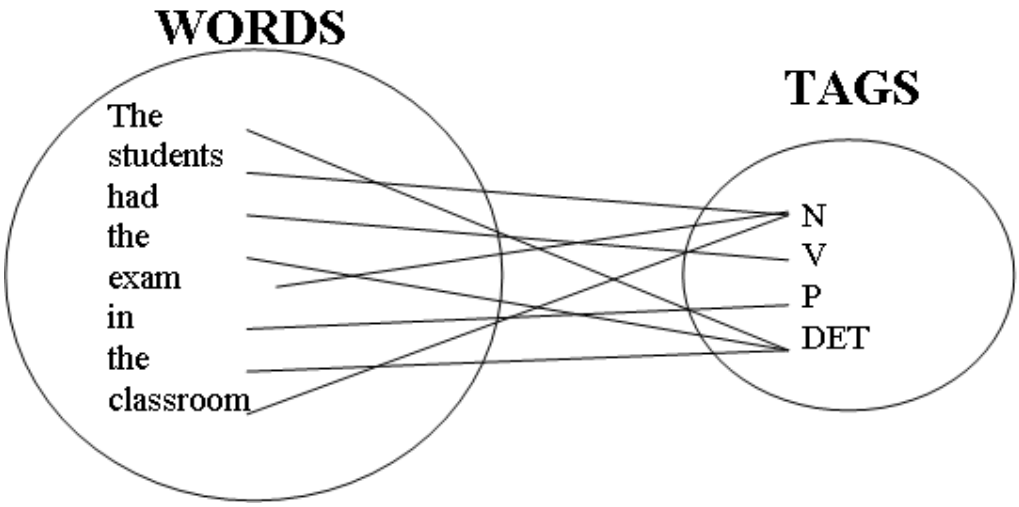

Figure 2.2 - Part-of-Speech tagging process

The thesis study has also used stemming, which is a suffix splitting method and that retrieves the base form for any given word. "Connect", "Connecting", "Connected", "Connection" and "Connections" words all have different suffixes and after these suffixes are removed the root term leaves as "Connect", as explained by Singh [15].

Another method is the phrase chunking, which is the process of extracting higher-level structure such as phrases created from pair of words. For example, in the sentence "Mike Jones will take the courses", there exists a proper noun phrase "Mike Jones", a verb phrase "will take" and a common noun phrase "the courses" as given by [13].

2.4 Machine Learning and Support Vector Machine

Gimenez et al. [16] defines Machine Learning (ML) as a subfield of Artificial Intelligence that produces various algorithms and methods in order to make computers "intelligent". It is also stated that, in addition to using data mining and statistics, ML also uses computational and statistical methods in order to collect a variety of information automatically from pure data. Some research areas for ML applications are exampled as natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics, brain-machine interfaces and cheminformatics, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing and robot locomotion.

Nakagawa et al. [17] says that in order to handle binary classification, an ML algorithm can be created on a feature vector space that is called "Support Vector Machine (SVM)". Moreover, they also claim that a training data set (2) is separated into two classes/regions by several hyperplanes (1) according to the equation (3).

$$w \cdot x + b = 0, \qquad w \in R^{L}, \; b \in R \tag{1}$$

$$\{(x_i, y_i) \mid x_i \in R^{L}, \quad y_i \in \{\pm 1\}, \quad 1 \leq i \leq \ell\} \tag{2}$$

$$y_i \, (w \cdot x + b) \geq 1 \tag{3}$$

SVM aims to find the optimal hyperplane, which maximizes the related margin that is the distance between the hyperplane and the nearest points (See Figure 2.3). This hyperplane gives the minimum expected test error and becomes a solvable quadratic programming problem according to Nakagawa et al. [17]. In the related study of Nakagawa et al. [17], it is also stated that the sign of discriminant function f(x) at (4) and (5), may give an appropriate label value y for each test example x:

$$f(x) = w \cdot x + b \tag{4}$$

$$y = \mathrm{sgn}\,(f(x)) \tag{5}$$

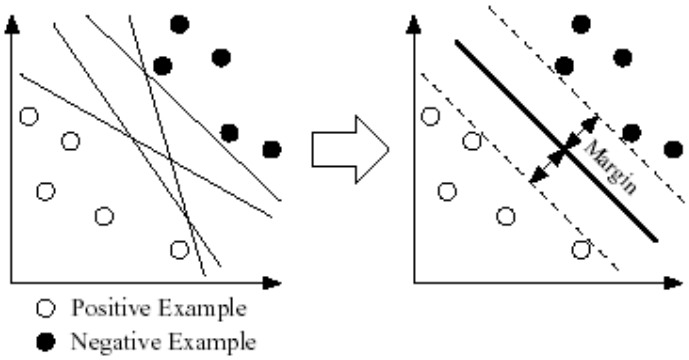

Figure 2.3 - Maximizing the margin

A dot product process between two vectors may be much expensive. Therefore, a *kernel function* may be introduced to avoid this numerical complexity drawback. By trying different kernel functions, various learning machines can be constructed in order to solve different problems [18]. Also, the training is not affected because of any huge or even infinite feature space dimensions [16]. SVM has many benefits for computational learning theory and it is very open and applicable to theoretical understanding and analysis according to Joachims [19].

2.5 Related Work

Software architects partition their applications due to several architectural concerns in a way that they can model, design, develop, test, and even maintain every part separately and supervise the development easily. Behavioral aspects of an information system are critical in this partitioning. But, non-functional requirements are more significant for realizing the problem domain and supervising the design of solution domain. However, there is no mutual understanding to localize the architectural concerns in the problem domain and a common roadmap to extract them from the NFRs list [1].

A number of studies have been presented to define precise methods (automated or semi-automated as well) for identifying the NFRs and achieving early aspects. For example, Cleland-Huang et al. [20] introduced an information retrieval based approach where detection and classification of NFRs can be automated, and this enables system level constraints to be considered and incorporated into early architectural designs. Steele et al. [21] portrayed an automatic speech recognition technique for capturing the non-functional requirements spoken by the stakeholders at open meetings and interviews during the requirements elicitation process. Extended problem frames, allowing architectural structures, services and artifacts to be considered as part of the problem domain are introduced by Hall et al. [22]. It is studied how this extension may enhance the applicability of problem frames in relating requirements and architectures. Baniassad et al. [23] gave an overview of how to exploit aspects earlier in software development life cycles, such as during requirements gathering and elicitation.

According to the aim of bringing NFRs into software architecture, some previous studies have been carried out. Chung et al. [24] presented their own structure "NFR Framework" in order to categorize NFRs, then keep them in "Softgoal Interdependency Graphs" and finally classify them into "Correlation Catalogues". Resulting catalogues show the effects of architectural designs on miscellaneous NFR softgoals. Later, the software architect selects among alternatives by evaluating the tradeoffs. Nevertheless, this study [24] is only an example for using the framework in an abstract level.

Another idea is reflecting NFRs in UML Diagrams according to Cysneiros et al. [9, 25]. They used the Chung's NFR Framework in addition to the tool LEL (Language Extended Lexicon) in order to pick keywords from plain texts and using these keywords while constructing NFR graphs. Then, according to these graphs, some attributes, classes etc are added into the UML diagrams until desired non-functionality is satisfied.

The use of NLP techniques in different phases of software engineering can be found in the literature. For example, some methods for mapping natural language elements to object-oriented concepts in requirements elicitation are discussed by Li et al. [26]. One such approach, namely Early-AIM to identify aspects in requirements is introduced by Sampaio et al. [27, 28] and evaluated by Chitchyan et al. [29]. This approach utilizes corpus-based NLP techniques to enable the identification and modeling of early aspects in a semi-automated way. The proposed technique describes how unstructured sources of requirements can be automatically mined to help the requirements engineer in identifying and building a structured aspect-oriented model of the requirements.

2.6 Remarks

This thesis tries to identify problem domain concerns in architectural modeling of software systems in an automated manner. On this road, a Support Vector Machines algorithm is used to relate NFRs to classified "architectural concerns" and Natural Language Processing techniques as part-of-speech tagging, phrase chunking and stemming are applied to fragment the plain NFR texts under the supervision of domain experts. Instead of others the SVM algorithm is used since it is known to have a good generalization performance. Moreover, SVM can handle a large number of features and hardly overfit. SVM is also strongly capable for extracting the relevant discriminatory information from the training data and it can employ different kernels involving many execution parameters. Hence, this makes finding the best solution extensive according to Jonsson et al. [30]. For the NLP part, LingPipe and Stanford libraries are used since they are one of the mostly-known NLP libraries and both are easy to integrate.

CHAPTER 3

THE PROPOSED APPROACH

Linking the requirements engineering process with architectural modeling is not straightforward. Especially, NFRs are extremely hard to capture and associate with the components and connectors of an architectural model. This thesis introduces a systematic approach to identify the NFRs by using NLP techniques as part-of-speech tagging, phrase chunking and stemming, and to associate the NFRs with quality attributes and architectural aspects by using ML. The approach is called as "Non-Functional Requirements to Architectural Concerns (NFR2AC)", which empowers the system architects to construct the Utility Concern Spaces (a matrix for the correlation of Architectural Aspects and Quality Attributes) from the NFRs expressed in plain text.

3.1 Motivation

Software architecture modeling is expected to relate the architectural concerns of problem domain to the running components and connectors of the solution domain. However, this mapping is not trivial, and it may surprisingly end up with problem domain concerns tangled in and scattered through the solution domain. In order to guide the software architect throughout this mapping, an inclusive architectural modeling approach has been proposed in [5], which identifies problem domain in "Utility Concern Spaces" by correlating the "Architectural Aspects" and "Quality Attributes", and solution domain in "Architectural Concern Spaces" by correlating the "Architectural Tiers" and "Architectural Views" depicted in Figure 3.1.

The original roadmap given in Figure 3.1 starts with the identification of quality requirements, which aims at determining the "architectural aspects" and "quality attributes" from NFRs of an application. In the original work [5], the approach did not propose a method to identify the architectural concerns from NFRs, but put the

way for mapping quality attributes to architectural aspects once they have been extracted from the NFRs. The proposed approach given for here, which is based on the research paper [1] attempts to systematize this extraction process using NLP and ML techniques. Thus, it covers only the parts encircled by the ellipses in Figure 3.1.

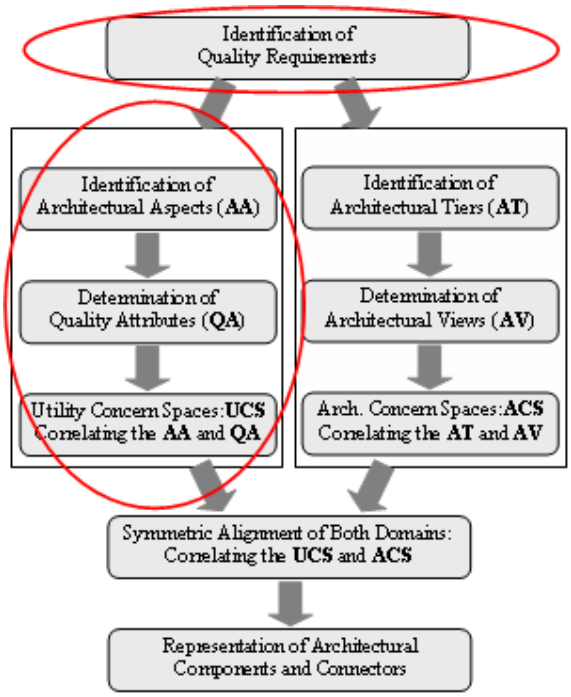

Figure 3.1 - Architecture modeling approach

The method proposed here demands on successive application of NLP and ML techniques for extracting the architectural concerns from NFRs, and training the mappings of quality attributes to architectural aspects. This way of handling NFRs in an automated environment refrains the system architects from having in-depth domain knowledge for architectural modeling of an application [1].

3.2 Definition of Architectural Concerns

Architectural Aspects (AA) and Quality Attributes (QA) constitute "architectural concerns" of the problem domain. In order to facilitate the NFR2AC approach with automated tool support, an initial taxonomy for AA and QA should be constructed. The know-how from several Web-based transactional applications at Cybersoft[1] has formed the taxonomy of Architectural Aspects. A limited snapshot of AA taxonomy is given in Table 3.1.

Table 3.1 - Taxonomy of "Architectural Aspects"

1. PRESENTATION ASPECTS	
1.1 Online Help Support	1.10 Browser-Based Use Support
1.2 Adherence to GUI Standards	1.11 Screen Layout Design
1.3 Inactivity Auto Logout Support	1.12 Status Bar Support
1.4 User Alerting	1.13 Locale Support
1.5 Input Validity Assurance	1.14 Printer Spool Support
1.6 Business Terminology Compliance	1.15 Login Screen Support
1.7 Multi Lingual Support	1.16 Context Sensitive Help Support
1.8 Reporting Format Support	1.17 Progress Indication
1.9 HTTP Support	
2. DATA ASPECTS	
2.1 Data Availability	2.4 Data Redundancy
2.2 Data Integrity	2.5 Data Capacity
2.3 Data Consistency	2.6 Data Uniqueness
3. INTEGRATION & COMMUNICATION ASPECTS	
3.1 Large Data Sets Limitation	3.3 System Integration Infrastructure
3.2 Adherence to Secure Integration Policies	3.4 Clustering & Fail Over Support
4. DEVELOPMENT ASPECTS	
4.1 Adherence to Framework Use Policies	4.4 Automated Functional Testing Support
4.2 Adherence to Coding Standards	4.5 Parametric Strings & Label Support
4.3 Adherence to Pattern Standards	
5. SECURITY ASPECTS	
5.1 Data Confidentiality	5.5 Service Level Authorization
5.2 Non-Repudiation	5.6 User Authentication
5.3 Security Alerting	5.7 Integrated Security Support
5.4 Data Security	5.8 Session Management
6. OPERATIONAL ASPECTS	
6.1 Data Backup	6.5 Authorized Service Assignment
6.2 System Liability Accordance	6.6 Installation Guide Provision
6.3 Alerting	6.7 Users Guide Provision
6.4 Password Management	
7. MAINTENANCE ASPECTS	
7.1 Data Restore	7.3 Rollback Management
7.2 Bug Fix Management	7.4 Change Management
8. APPLICATION & COMPUTATIONAL ASPECTS	
8.1 Transaction Throughput	8.7 Component Reuse
8.2 Response Time	8.8 Platform Independency
8.3 Progress Metering	8.9 Transaction Management
8.4 Time Management	8.10 Load Balancing
8.5 System Availability	8.11 Unique ID Generation
8.6 Adherence to Business Rule Standards	

[1] Cybersoft Information Technologies: http://www.cybersoft.com.tr

Another classification should be done for QA but, this time, Larsson Quality Model [31] was adapted. A limited snapshot of QA taxonomy is depicted in Table 3.2. NFR2AC approach revealed that for each AA and QA almost similar phrases are used to state the NFRs. So, it is possible to extract corresponding AA and QA from NFRs automatically. In order to do this, a domain vocabulary has to be constructed. This approach uses NLP and ML techniques to form a domain vocabulary that learns the correspondence of NFRs to AA and QA.

Table 3.2 - Taxonomy of "Quality Attributes"

1. USABILITY	
1.1 Accessibility	1.4 Generality
1.2 Administrability	1.5 Operability
1.3 Understandability	1.6 Simplicity
2. PORTABILITY	
2.1 Mobility	2.3 Hardware Independence
2.2 Nomadicity	2.4 Software Independence
3. PERFORMANCE	
3.1 Accuracy	3.7 CPU Utilization
3.2 Footprint	3.8 Latency
3.3 Responsiveness	3.9 Transaction Throughput
3.4 Scalability	3.10 Concurrency
3.5 Schedulability	3.11 Efficiency
3.6 Timeliness	
4. MAINTAINABILITY	
4.1 Flexibility	4.5 Upgradeability
4.2 Evolvability	4.6 Expandability
4.3 Extensibility	4.7 Data Consistency
4.4 Modifiability	4.8 Version Consistency
5. INTEGRABILITY	
5.1 Adaptability	5.5 Heterogeneity
5.2 Composability	5.6 Integrability
5.3 Interoperability	5.7 Audibility
5.4 Openness	5.8 Completeness
6. DESIGN	
6.1 Conciseness	6.6 Analyzability
6.2 Correctness	6.7 Modularity
6.3 Testability	6.8 Reusability
6.4 Traceability	6.9 Configurability
6.5 Coherence	6.10 Distributeability
7. DEPLOYABILITY	
7.1 Ease of Creation	7.3 Confidentiality
7.2 Availability	
8. DEPENDABILITY	
8.1 Integrity	8.5 Security
8.2 Maintainability	8.6 Cost
8.3 Reliability	8.7 Projected Lifetime
8.4 Safety	
9. BUSINESS ORIENTATION	
9.1 Targeted Market	9.3 Affordability
9.2 Time To Market	9.4 Development Time

3.3 The Roadmap

The charted roadmap of NFR2AC approach is given in Figure 3.2. NFR2AC has three phases:

- Phase 1 tries to construct the domain vocabulary.
- Phase 2 trains the knowledge base for mapping the NFRs to architectural concerns.
- Phase 3 is the "testing phase" for the brand new mappings of NFRs to AA and QA for the target application.

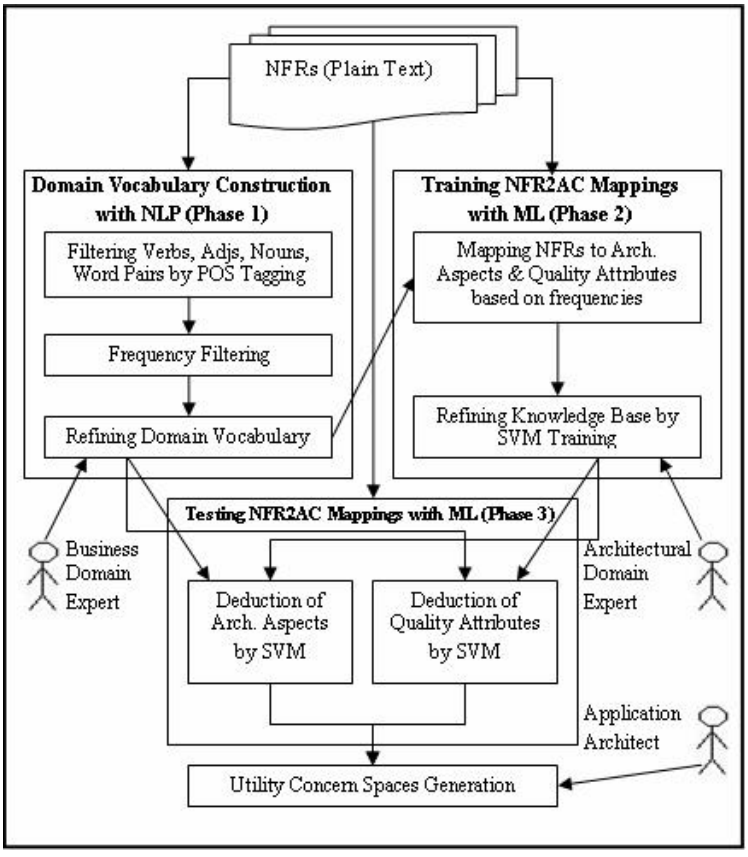

Figure 3.2 - Roadmap for NFR2AC approach

Phase 1 tries to construct the domain vocabulary, which accepts NFRs in plain text format and searches the "common phrases" using NLP techniques as part-of-speech tagging, phrase chunking and stemming. Business domain experts may help form a better domain vocabulary in this phase.

This approach assumes that if the raw NFR text is filtered, some of the less informative words can be eliminated and consequently only important phrases can be dealt with. For example, if the NFR is:

"The system will provide reporting facilities in suitable formats that are handled by normal end user computing equipment and applications. These end user computing equipment and applications are however, not considered as being part of the system."

The text will be inputted to NLP module, which will give the output of words together with their "Part of Speech" tags like verb, adjective, noun etc. Those words other than noun, adjectives and verb phrases like "the", "a", "can" will all be eliminated and the remaining ones will appear in root (stem) form with their frequencies. The formatted text will appear as:

system (2), end user (2), computing equipment (2), application (2), reporting facility(1), format(1) etc.

After the elimination, commonly used phrases are specified and the phrases with low frequencies are removed. By this way, the phrases belonging to the list of "important words of the NFR" are labeled with a unique "Phrase ID" and added to domain vocabulary. On the other hand, some phrases of an NFR may exist in the domain vocabulary. In this case, current IDs of the previously added phrases are found from the domain vocabulary before generating new IDs for new phrases. For the ID generation, a numerator table exists in NFR2AC database and the last generated ID is retrieved from this table incrementally. Let's assume that the last given ID is 270, and previously added phrases in domain vocabulary are *end user* –

phraseId: 56 and application – phraseId: 103. Then above structure will be as follows:

system – phraseId: 271
end user – phraseId: 56
computing equipment – phraseId: 272
application – phraseId: 103
reporting facility – phraseId: 273
format – phraseId: 274

After ID retrieval/generation process, first time seen phrases are inserted into the phrase table in database (See Table 3.3).

Table 3.3 - Phrase Table Content

CODE	VALUE
1	user
2	function
3	time
4	business
5	screen
6	end user
7	login
8	access

Domain vocabulary can be extended in two ways: first by repeating this process with more NFRs and second by the domain expert at any time. Domain expert can load new plain texts to the system containing the phrases that he/she wants to add to

the domain vocabulary. Additionally, domain expert can see the Cross Domain Occurrence information for each new phrase at this phase. Cross Domain Occurrence shows the previous occurrences of any phrase existing domains and previous projects, so this information may help domain expert choose new phrases to be added to the domain vocabulary.

Using the domain vocabulary constructed in Phase 1, Phase 2 trains the knowledge base for mapping the NFRs to architectural concerns with ML techniques using Support Vector Machine. The output of Phase 2 is an "SVM Vector" reflecting the mappings of NFRs to architectural aspects and quality attributes. Trainings may be repeated to improve the aptitude of repositories for more effective automated mappings. In this work, the system is trained with several mappings of NFRs to AA and QA collected from several Web-based transactional applications implemented at Cybersoft.

The aim of Phase 2 and Phase 3 (See Figure 3.2), is deducing the AA and QA corresponding to each NFR using SVM. This phase has two "operation modes": "training" and "testing" as all ML methods suggests. Phase 2 is known as the "training mode" where NFRs with the corresponding architectural aspect(s) and quality attribute(s) are provided to the training phase. In the beginning of Phase 2, NFR plain texts are loaded to the system and POS tagging, filtering, frequency calculation are done in the same way like in Phase 1. The main goal here is finding the key phrases in NFR sentences, and then constructing the SVM training vector with their phrase IDs. Domain vocabulary can be also constructed at this time. First time seen phrases are added to phrase table (See Table 3.3) and then NFRs are added to the NFR table (See Table 3.4). Finally, the information of "which NFRs contain which phrases" is put into the relation table (See Table 3.5).

Table 3.4 - NFR Table Content

PROJECT	NFR ID	NFR VALUE
Auto Comp.	1	Training material will be made available to the …
Auto Comp.	2	Due to the nature of the used technology…
Bank ABC	1	The system will provide online access to training…
Bank ABC	2	The operation of the system will require the users…

Table 3.5 - NFR-Phrase Relation Table Content

PROJECT	NFR ID	PHRASE	FREQUENCY
Auto Comp.	1	Training material	1
Auto Comp.	1	available	1
Auto Comp.	2	technology	1
Bank ABC	1	online access	1
Bank ABC	2	operation	1

The NFR sentence(s) are filtered to keep those phrases in the domain vocabulary and a training vector is constructed with IDs and frequencies of the filtered phrases. For the same example used in Phase 1, phrase IDs and frequencies are retrieved ib Phase 2 as:

"system – phraseId: 271 – frequency: 2, end user – phraseId: 56 – frequency: 2, computing equipment – phraseId: 272 – frequency: 2, application – phraseId: 103 – frequency: 2, reporting facility – phraseId: 273 – frequency: 1, format – phraseId: 274 – frequency: 1".

Then, the training vector becomes:

+1 271:0,2 56:0,2 272:0,2 103:0,2 273:0,1 274:0,1

"+1" states the occurrence of corresponding concern, and the rest of the string represents the "Phrase ID" and the "normalized frequency" for each phrase. This vector is sorted according to the ID labels in ascending order right before the SVM training session. This approach constructs a single vector for every provided architectural concern. SVM creates a model file for each architectural concern and the domain expert selects related architectural concerns after training vector has been constructed. The models of these architectural concerns are trained with the positive vector. Additionally, based on same vector elements a negative vector is also created and remaining models are trained with this negative training vector stating the absence of mapping with the other architectural concern. SVM model files are kept in a file system and one individual model file is created for each architectural concern. The models are named as ("model_" + (asp/qa) + Concern_id). Concern_id is given initially by the system due to the tables of taxonomy introduced before (e.g. Concern_id of Data Redundancy is 2.4). An SVM model example is given in Appendix B.1. NFR to architectural concerns relation is also kept in database as shown in Table 3.6.

Table 3.6 - Training Data Table Content

PROJECT	NFR ID	MAPPING TYPE	OID MAPPING
Auto Comp.	1	Architectural Aspect	1.7
Auto Comp.	1	Architectural Aspect	2.4
Auto Comp.	1	Quality Attribute	3.4
Auto Comp.	2	Quality Attribute	4
Bank ABC	1	Quality Attribute	5.6
Bank ABC	1	Architectural Aspect	1.1
Bank ABC	2	Architectural Aspect	2.2
Bank ABC	2	Architectural Aspect	3.1

After the positive and negative training vectors are constructed, these vectors are inserted into model data table given by Table 3.7 in the NFR2AC database.

Table 3.7 - Model Data Table Content

PRJ	NFR ID	MAP. TYPE	OID MAPPING	LBL	DIMS	VALUES
Auto C.	1	AA	1.7	1	1,7,15...	0.2,0.5,0.1,0.7...
Auto C.	1	AA	2.4	-1	1,7,15...	0.2,0.5,0.1,0.7...
Auto C.	1	QA	3.4	1	2,8,34...	0.3,0.1,0.7,0.7...
Auto C.	2	QA	4	1	5,7,75...	0.1,0.4,0.4,0.9...
Bank	1	QA	5.6	1	1,9,55...	0.3,0.5,0.3,0.2...
Bank	1	AA	1.1	-1	1,9,55...	0.3,0.5,0.3,0.2...
Bank	2	AA	2.2	1	2,3,11...	0.9,0.3,0.3,0.2...
Bank	2	AA	3.1	1	4,7,65...	0.6,0.6,0.1,0.5...

In Table 3.6 and 3.7, MAPPING_TYPE column specifies the type of architectural concern, whether it is an architectural aspect or a quality attribute. Additionally, OIDMAPPING column specifies which architectural aspect or quality attribute is having the same value according to the mapping type mentioned before as “Concern_id”. System initially stores distinctive IDs for each architectural concern. LABEL, DIMS1-2-3, VALUES1-2-3 columns store the training vector elements created.

Finally, Phase 3 is for the brand new mappings of NFRs to AA and QA for the target application. In ML terminology, this phase is known as the “testing phase” and its output is the Utility Concern Spaces formed by correlating AA and QA [5]. Application architects are free to adjust automated mappings, which can be

forwarded back to Phase 2 for further training. Continual training of the knowledge base from diverse business domains will improve the success of NFR2AC approach by forming more intelligent repositories.

During the testing phase (Phase 3), as new NFRs appear, each one is filtered similar to the training phase. Corresponding architectural concerns need to be found with the help of training vectors provided by Phase 2. Let's assume that our testing NFR is:

"The system will be web based and will therefore be accessible to the user using a standard installed and controlled web browser."

According to the methods previously introduced, the output of NLP module becomes:

web – phraseId: 312
standard – phraseId: 564
browser – phraseId: 9
web browser – phraseId: 274
system – phraseId: 181
user – phraseId: 59
accessible – phraseId: 77

After the POS tagging and elimination processes, another elimination method is executed for the phrases that are not in the current domain vocabulary. Since they are not in the domain vocabulary, they cannot retrieve a matching phrase ID and they cannot take part in the SVM testing vector. Also, it is nonsense to test the knowledge base with a non-trained element. The each remaining one has the frequency value 1. As a result, an SVM vector for testing is constructed with the same way a training vector created.

For each architectural concern, testing function is called one by one for the constructed vector. For each architectural concern, also that means for each trained model file, SVM returns a decision function value for each aspect and quality attribute according to the given testing vector. In the training mode, label value becomes "-1" for negative examples, in contrast to the case that the label value becomes "+1" for positive values. These are the peak values. Therefore, the return value for SVM decision function is normalized to obtain a probability of the architectural concerns occurrences. If the value is less than or equal to "0%", then the probability of architectural concern occurrence is "%1". Similar to the minimum peak value, if the value is greater than or equal to "100%", then the probability of architectural concern occurrence is "99%". For the peak values, NFR2AC approach avoids to result "%0" and "%100" values, because SVM is not a highly confidential method for the default execution parameters. A small error possibility is released in the testing phase

.

Table 3.8 - Testing Data Table Content

PROJECT	NFR ID	MAPPING TYPE	OID MAPPING	VALUE
Auto Comp.	10	AA	1.7	-0.2330
Auto Comp.	10	AA	4.4	0.4430
Auto Comp.	11	QA	1.4	-0.1150
Auto Comp.	21	QA	4.5	0.9980
Bank ABC	14	QA	5	-0.6346
Bank ABC	15	AA	1.1	-0.4550
Bank ABC	22	AA	3.3	0.5630
Bank ABC	28	AA	2.3	0.6650

After SVM testing execution, results are saved in a database as the training data. One reason for this is generating and exposing the "Cross Domain Occurrence" information from the related database table. This information is necessary for the parts introduced before. The second reason is leaving a knowledge base for constructing the problem domain matrices for the UCS generation. Another benefit of this information might be calculating some other probabilistic values from the database as a future work. The testing results are inserted into the classified results table given in Table 3.8. Additionally, this table saves the SVM testing results in VALUE column in a non-normalized form.

After several training and testing sessions, a highly qualified knowledge base can be constructed. For the Utility Concern Spaces transformation, the correlation matrix can be constructed by relating architectural aspects and quality attributes. The cell values of this correlation matrix will be filled out by the information stored in the knowledge base (previous mappings learned from the several trainings of diverse business domains). In order to construct the problem domain matrix, those architectural aspects and quality attributes with more than 50% probability for any SVM testing session are selected. Then, these aspects and quality attributes are supplied to the problem domain matrix and the matrix cells are filled with correlation figures reflecting the encountered values of related architectural aspect and quality attribute. In order to calculate this value, a concern occurrence matrix is constructed according to the relation between each architectural aspect and quality attribute as follows:

Table 3.9 – Architectural Aspect/Quality Attribute Occurrence Matrix

	QA1	QA2	QA3
AA1	a	b	c
AA2	d	e	f
AA3	g	h	i

In the Table 3.9 occurrence numbers of architectural aspects and quality attributes in same NFRs in testing results are represented as “a, b, c, d, e, f, g, h, i” in a matrix. The cell value of UCS matrix for each architectural aspect and quality attribute pair is calculated according to this matrix. For example in UCS matrix, the cell value of (AA1, QA1) is calculated with the formula:

$$\text{Value} = ((a / a+b+c) + (a / a+d+g)) / 2 \quad (6)$$

After this value for each cell is calculated, it is scaled as being “Neutral”, “Strong (+)” and “Very Strong (++)”according to some peak values. Domain expert can update each cell after the automatic filling. The formula (6) gives an average weighting factor for QA and AA correspondence since quality attributes might crosscut the architectural aspects as well as the architectural aspects might crosscut the quality attributes.

When the tables get their final state under the domain expert control, resulting matrices can be exported to an XML file to be used by the other steps of the architecture modeling approach [5].

In the original work [5], UCS is constructed by correlating the quality attributes and the architectural aspects in a matrix, where the architectural aspects show the rows and quality attributes constitute the columns (See Figure 3.3). In the matrix, every correlation set (the cubes of upper illustration in Figure 3.3) embodies an individual utility concern space.

Similarly, ACS is constructed by correlating the architectural tiers and the architectural views in a matrix, where the architectural views show the rows and architectural tiers constitute the columns (See Figure 3.4). In the matrix, every correlation set (the cubes of upper illustration in Figure 3.4) embodies an individual architectural concern space.

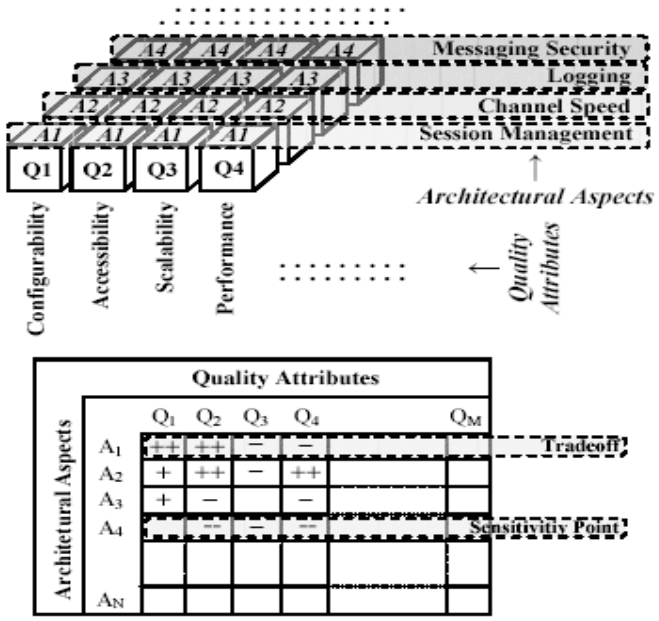

Figure 3.3 - Utility Concern Spaces

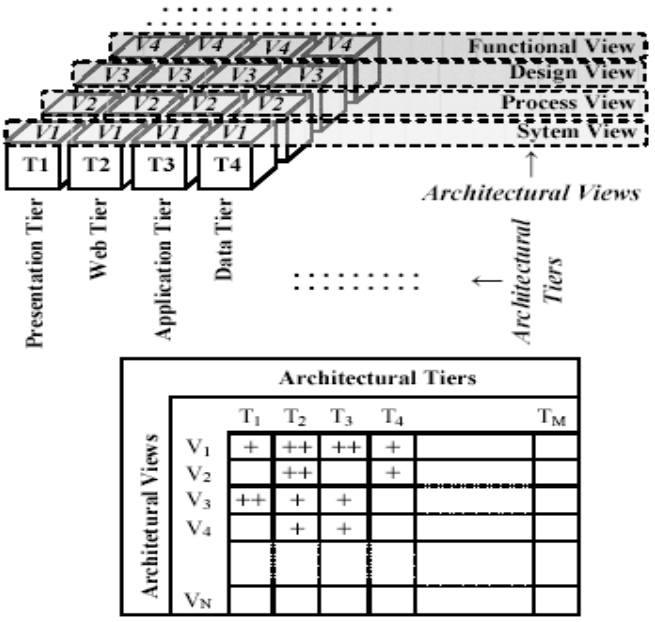

Figure 3.4 - Architectural Concern Spaces

The “+” or “-” symbol denotes a positive or negative correlation, and “++” or “--” symbol shows the stronger correlations. The coefficients play an essential role for identifying the “sensitivity points” and “tradeoffs” [5].

3.4 Remarks

The proposed approach is explained in this chapter. Taxonomies for architectural concerns are listed in the beginning of this section and subsequently NFR2AC Roadmap is introduced step by step. It consists of three main phases. The first step is constructing a domain vocabulary by selecting key phrases from plain texts. Then, by using this domain vocabulary, a knowledge base is constructed according to the NFRs – architectural concerns mappings specified by the domain expert. After obtaining a well-trained knowledge base, brand new mapping relations are searched over this knowledge base by loading NFR plain texts. Finally, after getting successful testing results, problem domain matrices are formed in an automatic manner in order to finding the Utility Concern Spaces that is the scope of the original paper [5] which this study aims to improve.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTATION

This chapter explains the implementation and experimentation details.

4.1 Implementation

In order to validate the proposed approach, an NFR2AC tool (an Eclipse plug-in) is developed to demonstrate the proposed steps of the related work.

4.1.1 Environment and Tools

NFR2AC tool uses some libraries for its sub-modules. For the "Domain Vocabulary Construction with NLP module" of the Phase 1, LingPipe[2] and Stanford[3] software are used as part-of-speech tagger, phrase chunker, stemmer utilities. On the other hand, for the ML module, SVM^{light} framework[4] and JNI Kernel Extension for SVM^{light} software[5] are used. For the relational database, MySQL[6], and for the O2R mapping, Hibernate API[7] is used.

4.1.2 NFR2AC Toolset

NFR2AC toolset consists of six tab pages, each one dealing with dedicated tasks. All six pages of NFR2AC are shown in the next six figures (See Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6).

---

[2] LingPipe NLP API, "Alias-i LingPipe Natural Language Processing Java Libraries", http://alias-i.com/lingpipe/index.html

[3] Stanford NLP API, "The Stanford Natural Language Processing Java Libraries", http://nlp.stanford.edu/index.shtml

[4] SVMlight, "SVM light, Support Vector Machine framework", http://svmlight.joachims.org/

[5] JNI Kernel Extension for SVM light software, http://www.aifb.uni-karlsruhe.de/WBS/sbl/software/jnikernel/

[6] MySQL Database, http://www.mysql.com

[7] Hibernate Object To Relational Mapping API, http://www.hibernate.org

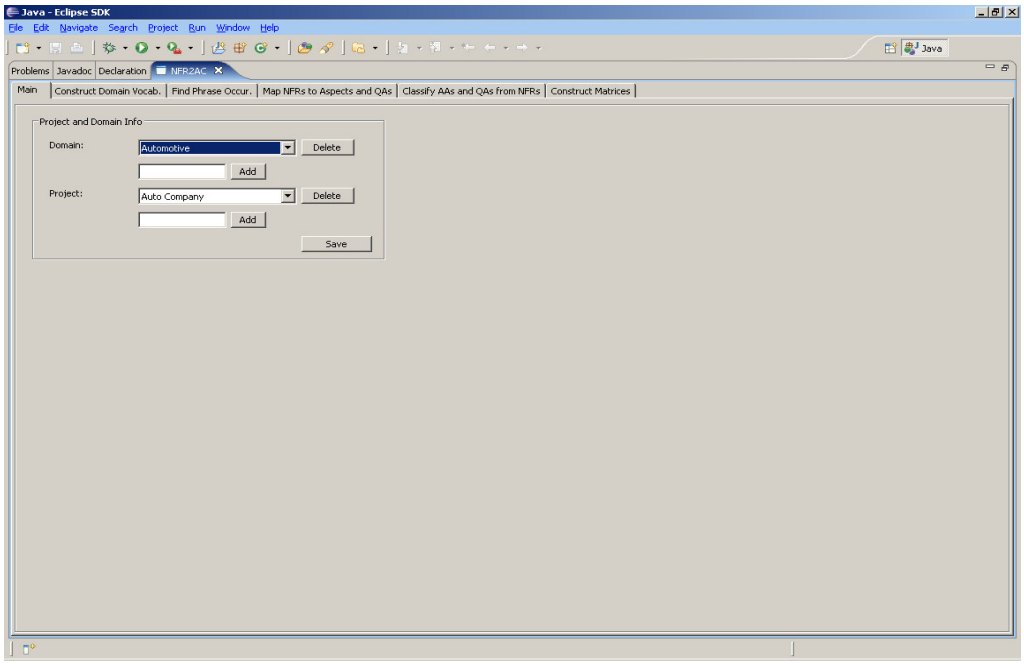

Figure 4.1 - Domain and Project Selection Tab

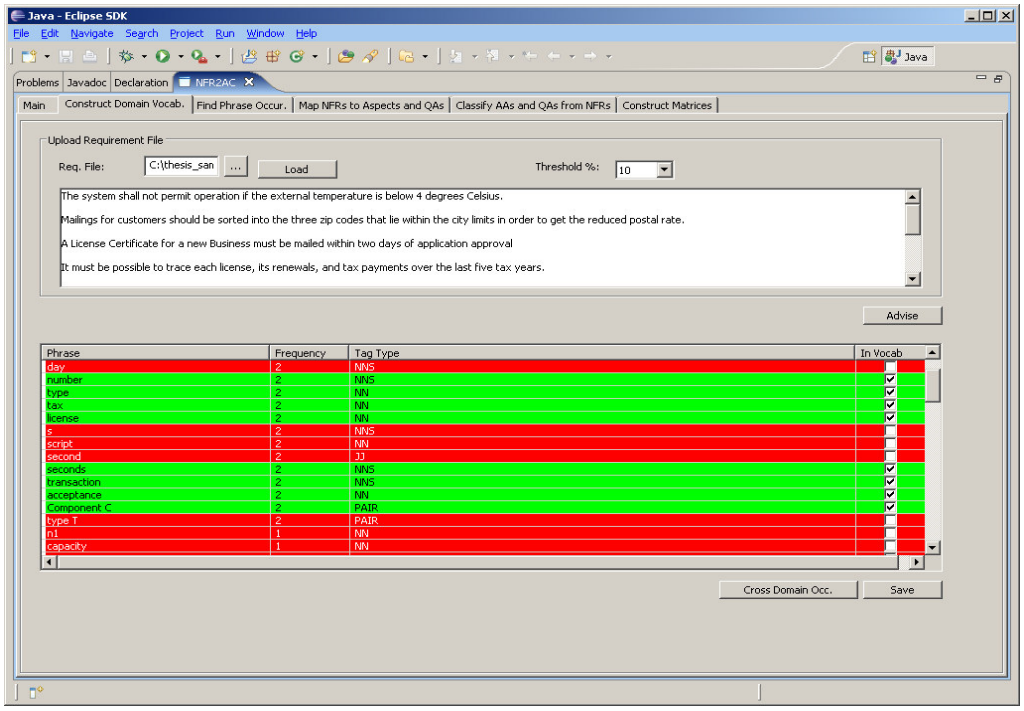

Figure 4.2 - Construct Domain Vocabulary Tab

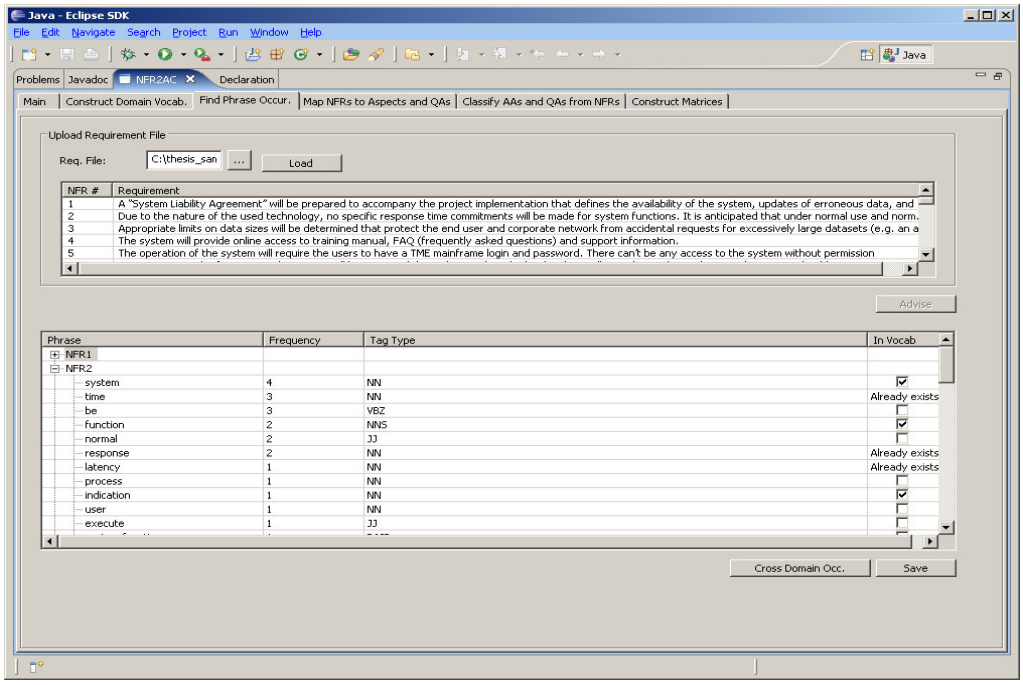

Figure 4.3 - Find Phrase Occurrences Tab

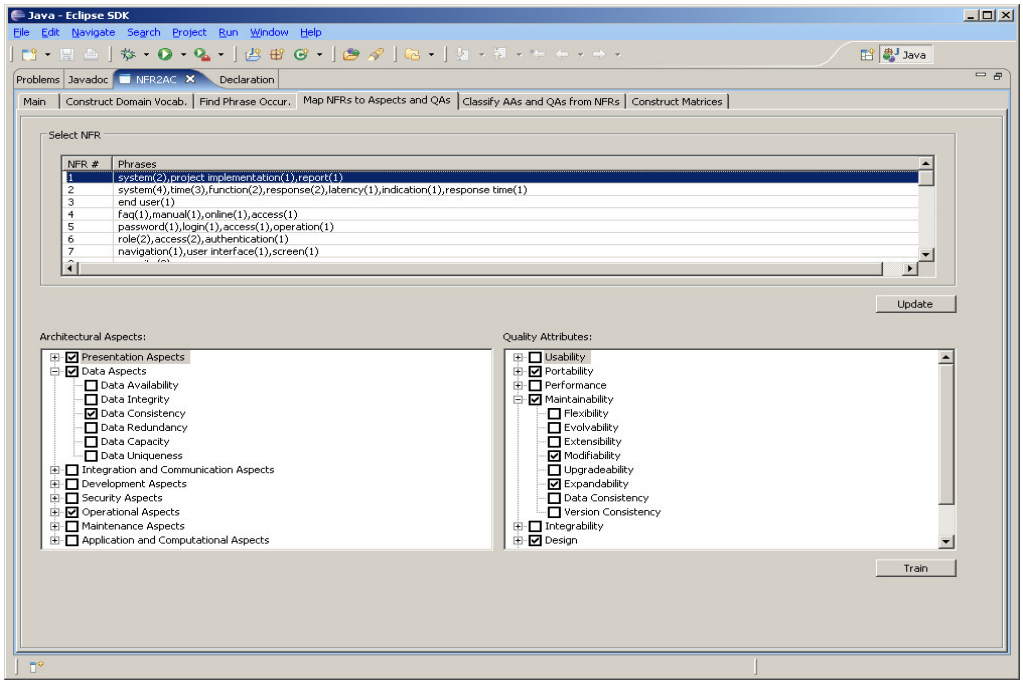

Figure 4.4 - Map NFRs to Architectural Aspects and Quality Attributes Tab

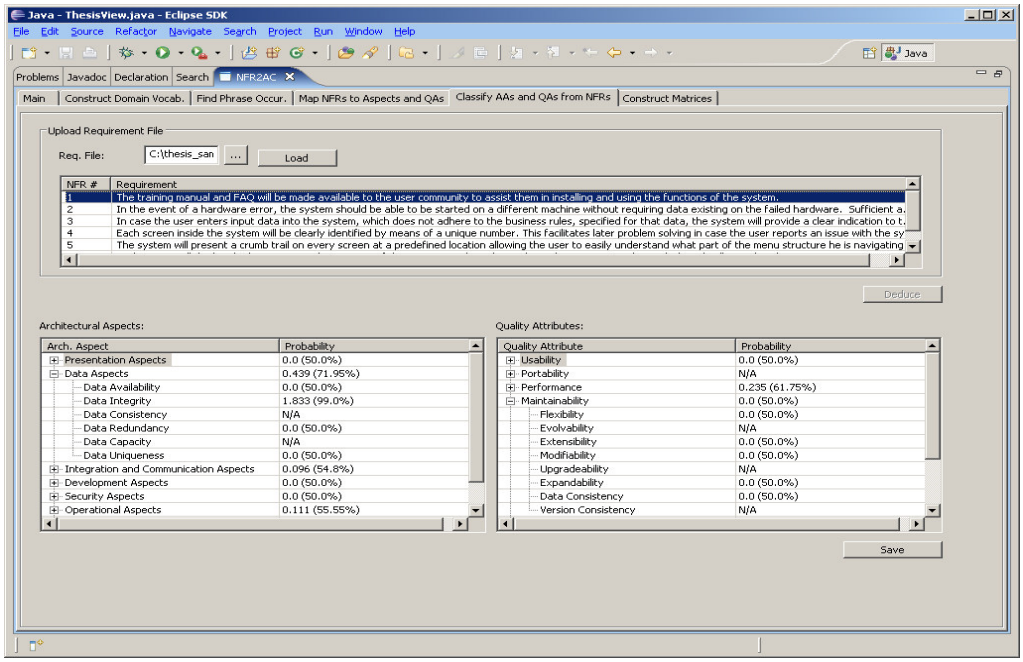

Figure 4.5 - Classify Architectural Aspects and Quality Attributes from NFRs Tab

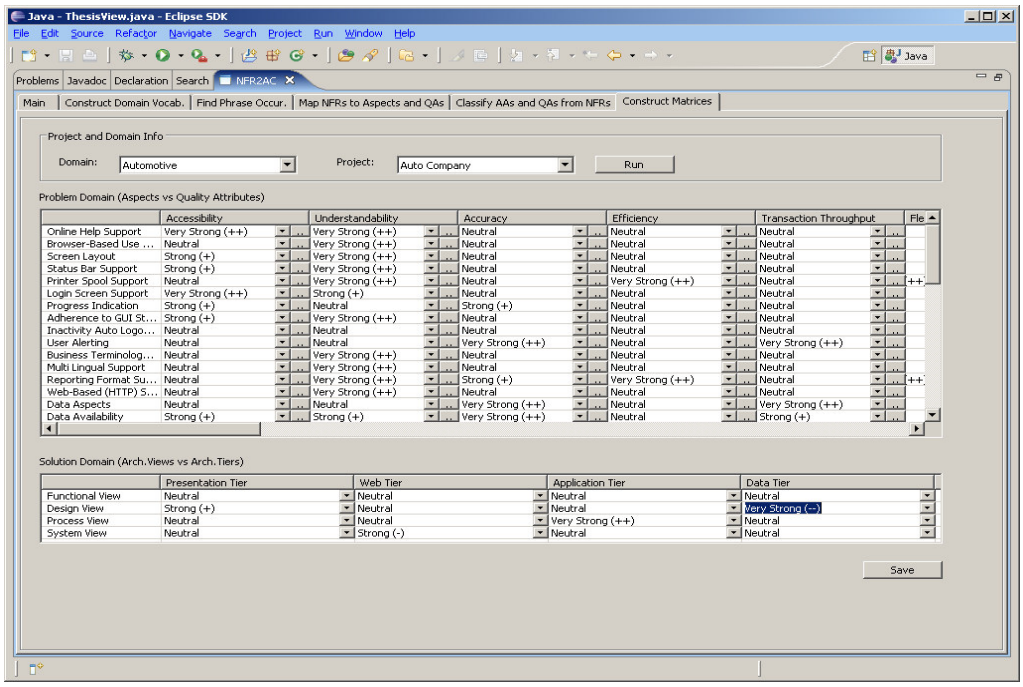

Figure 4.6 - Construct Matrices Tab

In the first tab, user is expected to select a “business domain” and associated “project”. Then, s/he uses the second tab and refines the domain vocabulary according to Phase 1 of the proposed approach. The third tab page (Figure 4.3) is designed to load the NFRs to the system in plain text form. Both in the second and third tab pages, phrases are also shown with the “cross-domain occurrence”, which reveals the occurrence of the related phrase in previous domain-project pairs with the probability values and the architectural concern relations in detail. This information guides the architects in choosing “common phrases”.

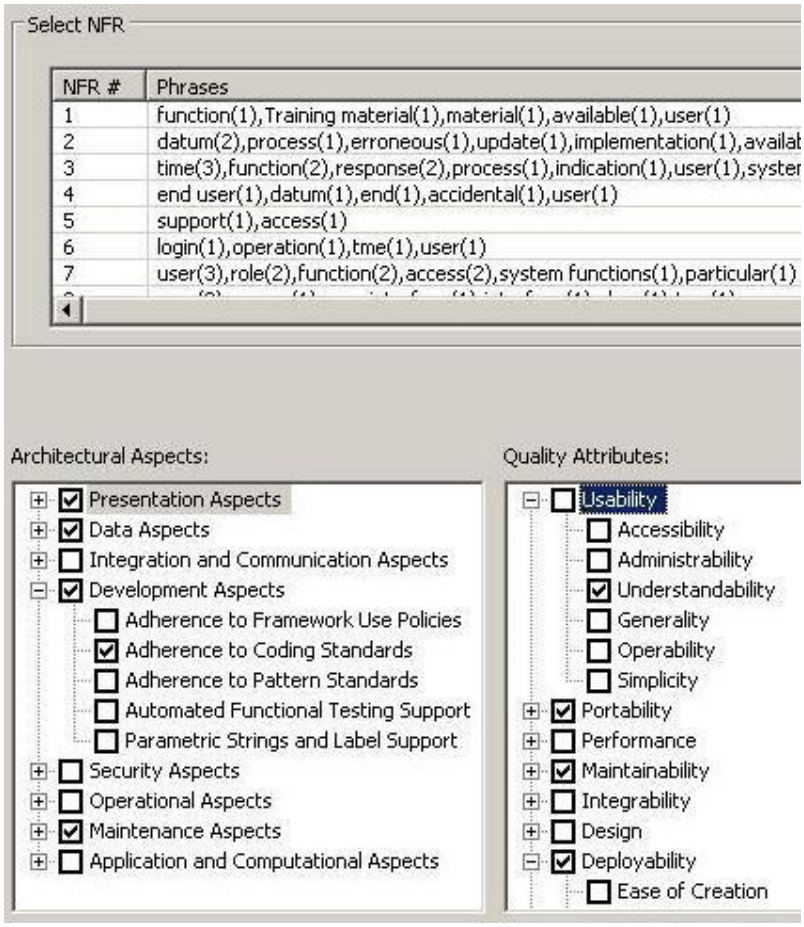

Figure 4.7 - Mapping of NFRs to Architectural Aspects and Quality Attributes

In the fourth tab (Figure 4.4), the domain vocabulary phrases are filtered and shown as "phrase-frequency list" for every NFR (e.g. Phrase1 (freq1), Phrase2 (freq2), Phrase3 (freq3), etc.), which forms the upper grid of the screen depicted in Figure 4.7. Then, user is expected to select the NFRs from the upper grid and associate with the AA and QA in the lower windows of Figure 4.7.

In order to commit the mappings, user presses the Save button, and SVM training session is completed. The training sessions can be repeated several times with different mappings of NFRs to architectural aspects and quality attributes, which improves the aptitude of knowledge base for future mappings of the targeted application.

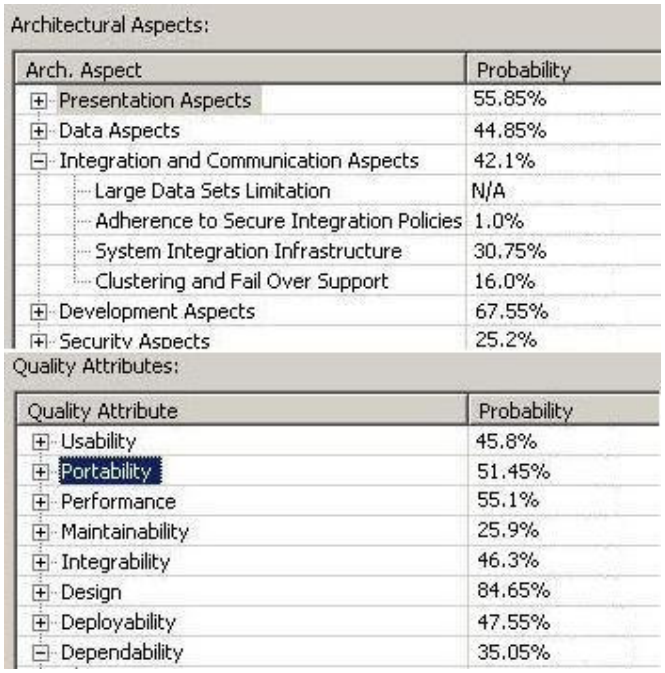

Figure 4.8 - Classifying Architectural Aspects and Quality Attributes

When SVM models are trained with enough mapping and training sessions, it is now time to test the SVM models and check whether SVM classification results are efficient enough. For this purpose, the fifth tab (Figure 4.5) is designed to load new NFRs to the system. Once the Deduce button is clicked, the system presents the occurrence probabilities of each architectural concern for every NFR as shown in Figure 4.8.

If the system has not any trained model for aspects or quality attributes, classification returns the "N/A (Non Applicable)" value. Finally, when the user clicks the Save button in the fifth tab page, classification results can be exported to an XML file.

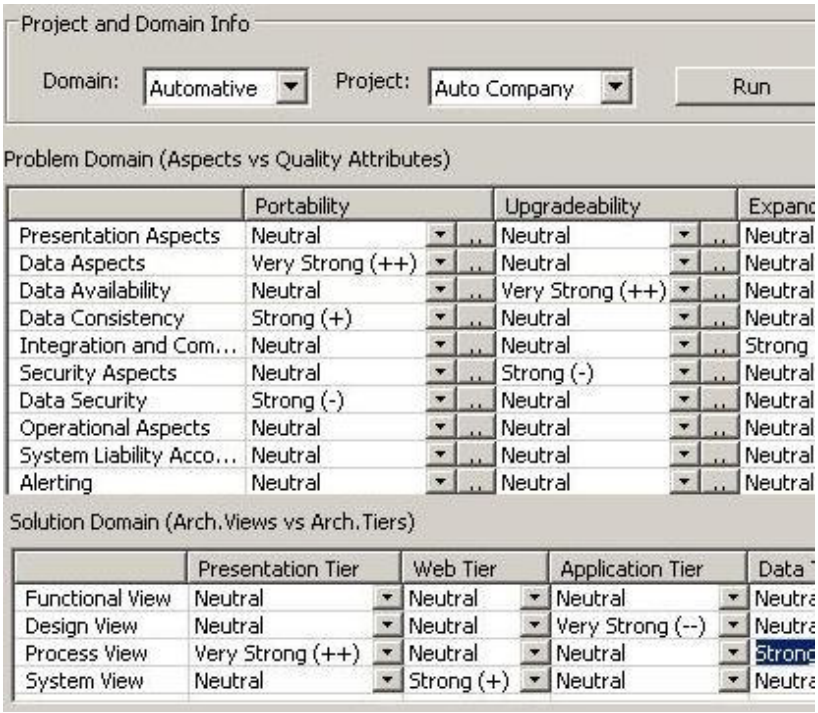

Figure 4.9 - Constructing the Utility Concern Spaces matrix

Once system has enough training and classification sessions, these observations and experimental results can be provided to the construction of Utility Concern Spaces matrix of the broader architectural modeling approach given in [5]. The sixth tab page (Figure 4.6) has been designed to manage this mapping. In this tab page, the user can construct the problem domain matrix (upper grid) and solution domain matrix (lower grid) regardless of any business domain and associated project (See Figure 4.9). Construction of the "Solution Domain" of modeling approach given in [5] is out of scope for this work, but solution domain matrix can also be generated similarly.

For the Utility Concern Spaces transformation, the correlation matrix can be constructed by relating architectural aspects and quality attributes according to the previous mappings learned from the several trainings of diverse business domains. Then, these aspects and quality attributes are supplied to the problem domain matrix and the matrix cells are filled with one of the options from "Neutral", "Strong (+)", "Very Strong (++)" initially. "Strong" and "Very Strong" options can be initialized for only plus values. Negative values are out of scope and can be declared as a future work. Domain expert can update each cell after the automatic filling.

4.1.3 UML Diagrams

In this section, the UML diagrams will be given to explain the whole system in detail.

The first diagram is a use case diagram that can be seen in Figure 4.10. It gives a full picture summary for the NFR2AC system. There are three kinds of users who might use the system each are dealing with a different task. First type of user is the "Business Domain Expert". This user is responsible for constructing a domain vocabulary by loading NFR plain texts, selecting key phrases within the requirements, and adding them into the domain vocabulary. Second type of user is the "Architectural Domain Expert". This user deals with constructing a knowledge base. In order to do that, user loads NFR plain texts to the system, and trains the

system with NFR – AC mappings, by using the domain vocabulary. Third user type is the "Application Architect". This user tests the knowledge base by loading new NFR plain texts. Then, if results are efficient, user generates the "Utility Concern Spaces" from system, by using the knowledge base.

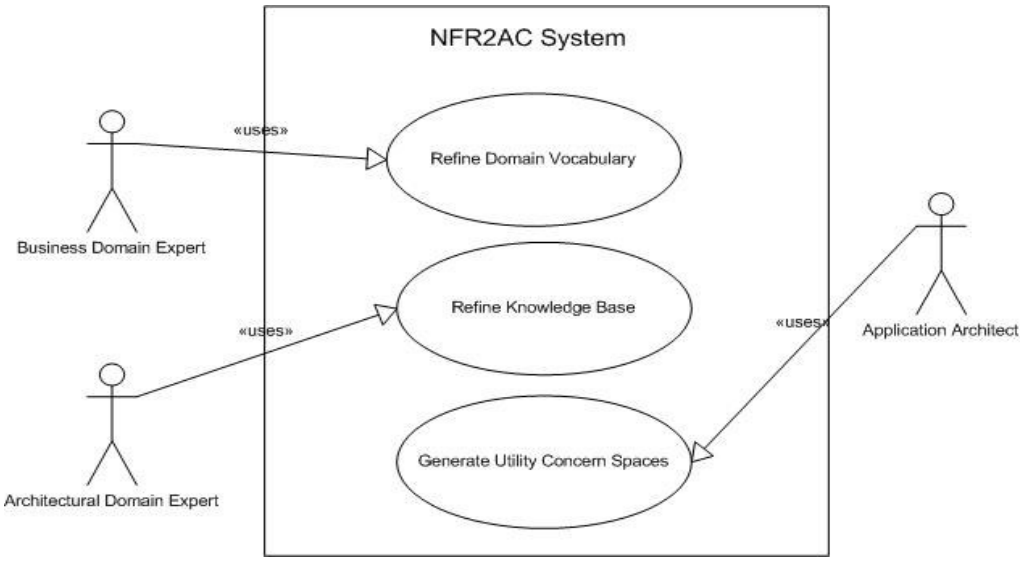

Figure 4.10 - Use case diagram of NFR2AC

The second diagram is the activity diagram (See Figure 4.11). This figure explains the process in an activity flow, which is shown in a static way in the use case diagram. In the activity diagram, process starts with refining domain vocabulary until it seems sufficient for the next step. Otherwise, refining process is repeated. When it seems that a sufficient domain vocabulary is constructed, with mapping, training and testing sessions; "Refine Knowledge Base" process is applied. In order to get sufficient results in this step, the first step can be repeated too. Then, if the knowledge base is refined sufficiently, UCS generation step has begun. If results are

good enough, the system goal is accomplished. Otherwise, all previous steps are repeated until the system outputs better results.

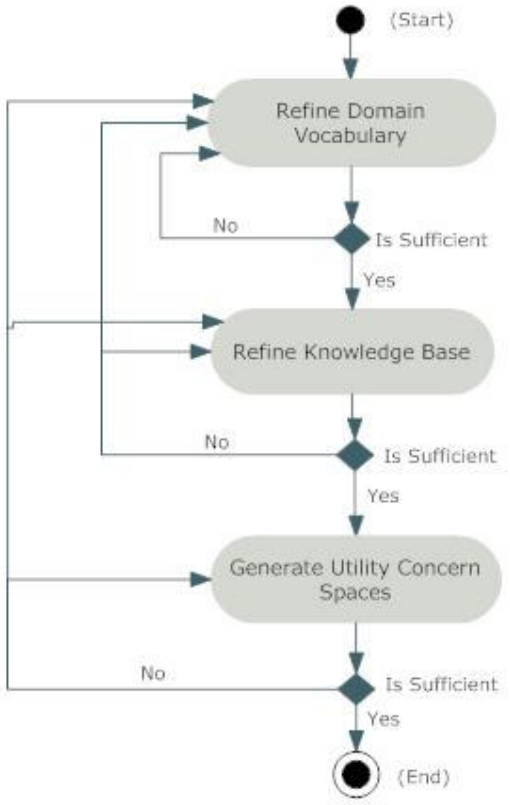

Figure 4.11 - Activity diagram of NFR2AC

In order to describe the whole process in detail, three sequence diagrams are given in the following three figures, each are explaining an individual phase of the system. First sequence diagram (See Figure 4.12) shows the sequential procedure of Phase 1, which is defined as “Domain Vocabulary Construction with NLP” in the current roadmap (See Figure 3.2).

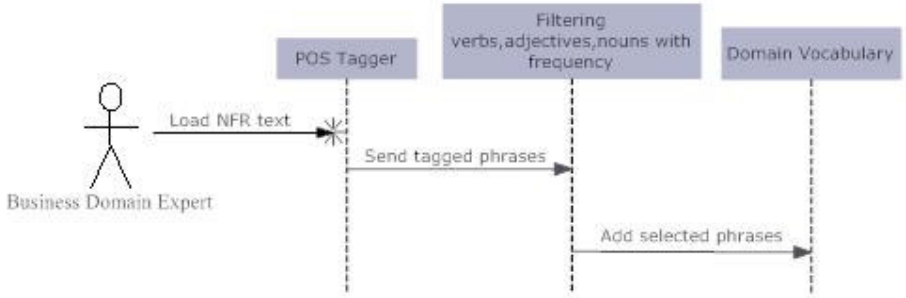

Figure 4.12 - Sequence diagram of Phase1

"Business Domain Expert" loads an NFR plain text to the system. Then, POS tagger tags each word and phrase. After that, phrases that are different than verbs, adjectives and nouns are eliminated. Remaining phrases are sorted by frequency in descending order. Finally, user selects important phrases from that list and adds them to the domain vocabulary.

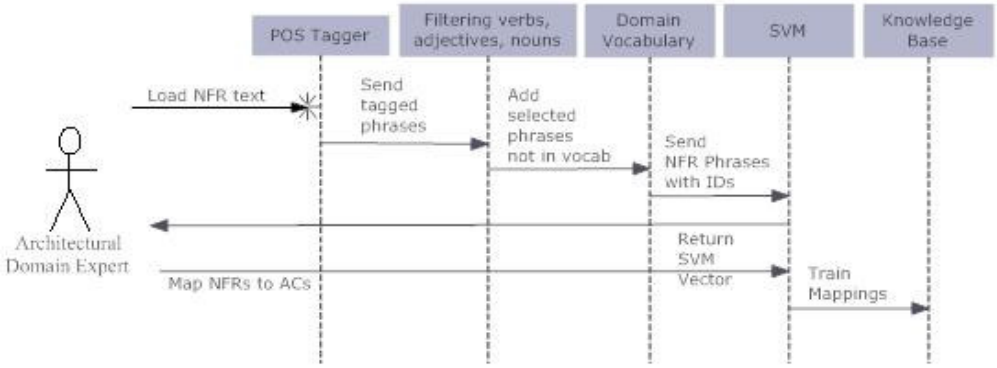

Figure 4.13 - Sequence diagram of Phase2

The second sequence diagram, shown in Figure 4.13, describes the second phase of the system defined as “Training NFR2AC Mappings with ML”. “Architectural Domain Expert” loads an NFR plain text to the system at the beginning. Then, as in Phase 1, phrases that are different than verbs, adjectives and nouns are eliminated. At this moment, user can add new phrases that are not added to the domain vocabulary before. Then, by getting all phrase IDs from domain vocabulary, SVM training vector is constructed. After user does mappings, knowledge base is trained with constructed training vector according to the mapping information.

The third sequence diagram, shown in Figure 4.14, describes the third phase of the system defined as “Testing NFR2AC Mappings with ML”. “Application Architect” loads an NFR plain text to the system. Then, as in previous phases, phrases that are different than verbs, adjectives and nouns are eliminated. Then, by getting all phrase IDs from domain vocabulary, SVM training vector is constructed due to the key phrases exist in the NFRs. After knowledge base is tested with constructed training vector, user can generate UCS matrix due to the sufficient test results.

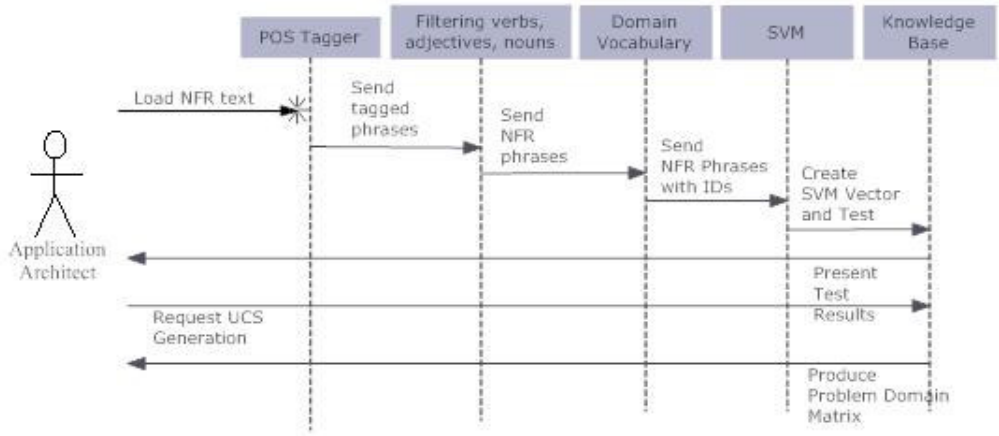

Figure 4.14 - Sequence diagram of Phase3

Figure 4.15 illustrates the package diagram of the NFR2AC toolset.

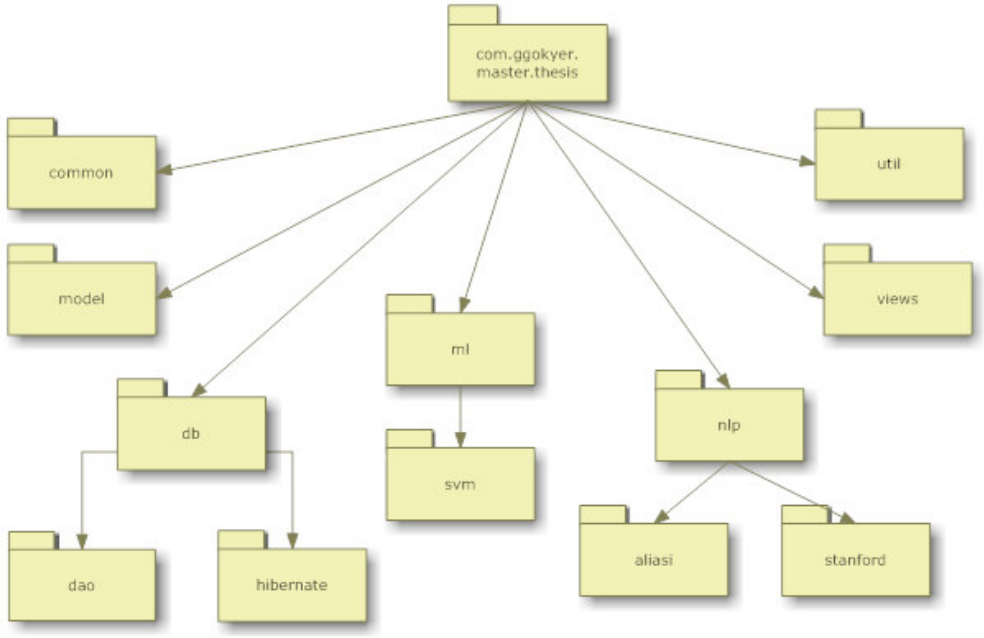

Figure 4.15 - Package diagram of NFR2AC

- com.ggokyer.master.thesis

This package includes all the other packages.

- com.ggokyer.master.thesis.common

This package includes the factory classes for the screen components.

- com.ggokyer.master.thesis.db

This package includes the database related packages,

- com.ggokyer.master.thesis.db.dao

This package includes the data access object classes for the O2R mapping.

- com.ggokyer.master.thesis.db.hibernate

This package includes the Hibernate configuration XML files.

- com.ggokyer.master.thesis.ml.svm

This package includes the Support Vector Machine related classes for Machine Learning actions.

- com.ggokyer.master.thesis.model

This package includes the data transfer object classes between the screens and the core classes.

- com.ggokyer.master.thesis.nlp

This package includes the NLP related packages.

- com.ggokyer.master.thesis.nlp.stanford

This package includes the NLP Stanford library classes.

- com.ggokyer.master.thesis.nlp.aliasi

This package includes the NLP Aliasi LingPipe library classes.

- com.ggokyer.master.thesis.util

This package includes main utility classes that the system needs.

- com.ggokyer.master.thesis.views

This package includes the plug-in screen classes.

4.1.4 Database Model

In this section, tables in the NFR2AC database will be listed with the detailed information about the mission for each one.

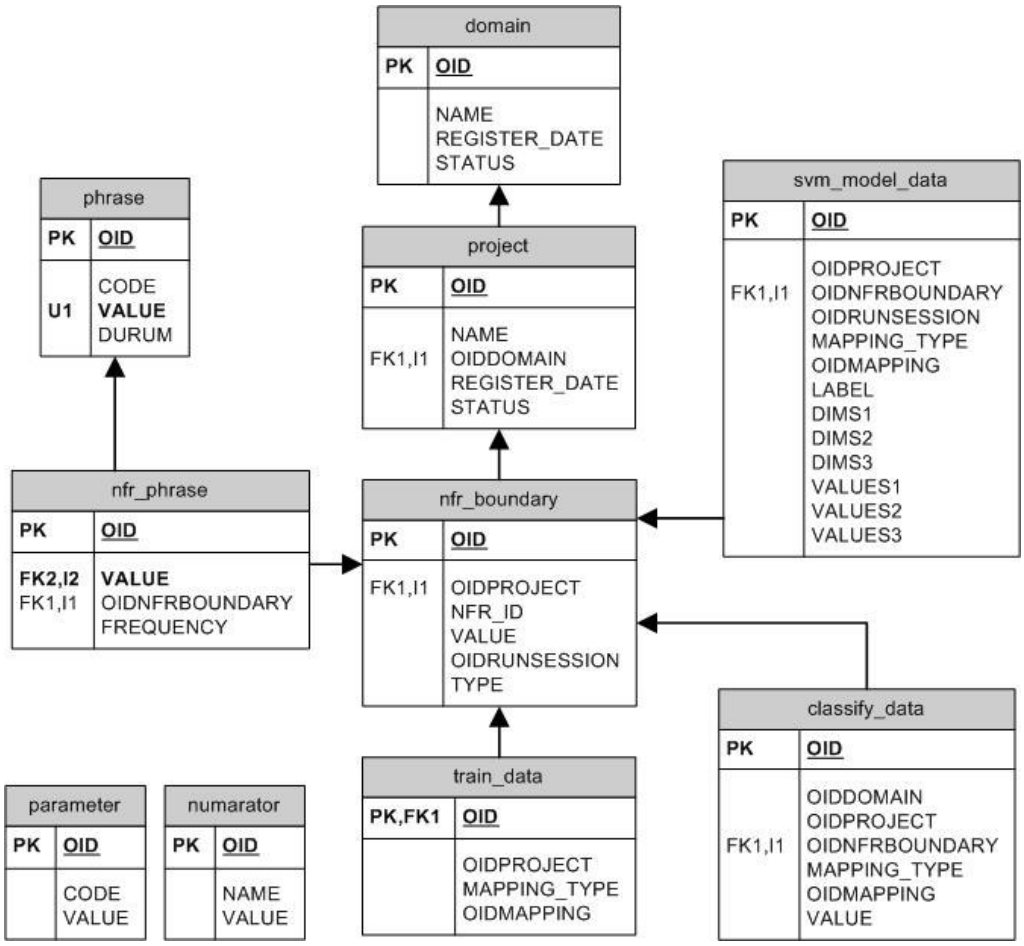

Figure 4.16 - NFR2AC Database Model

The database model of NFR2AC is shown in Figure 4.16. Tables of NFR2AC are:

- domain

  This table holds the domain information inserted in the first tab page by user.

- project

This table holds the project information inserted related to a domain in the first tab page by user.

- phrase

This table holds the phrases according to constructing the domain vocabulary.

- nfr_boundary

This table holds the NFR sentences imported by user in order to train or test the system.

- nfr_phrase

This table holds the relation of NFR sentences and key phrases within each NFR sentence.

- train_data

This table holds the training data that is the relation between NFRs and ACs mapped by user.

- classify_data

This table holds the testing results data that is created due to the NFR sentences loaded by user.

- svm_model_data

This table holds the SVM training vectors (label, dims, and values) that are created due to the NFR Sentences, in order to train the knowledge base.

- numarator

This table holds the numarator value for the phrase code. Every phrase takes a code incrementally from this numarator table, just before inserted to the domain vocabulary.

- parameter

This table holds the parametric values that are user by the system.

4.1.5 Usage of Software

NFR2AC tool is designed as an Eclipse plug-in. Hence, it is opened within the Eclipse IDE. From the Window menu, "Show View > Other" menu item is pressed in order to see the installed Eclipse views (See Figure 4.17).

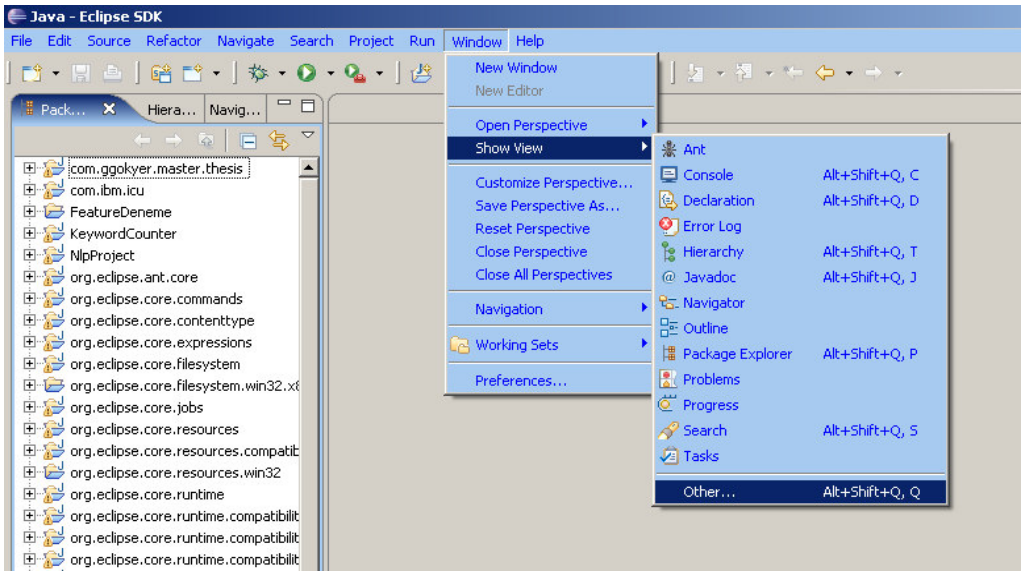

Figure 4.17 - Running the NFR2AC

NFR2AC tool is added to the Eclipse views menu with a title “NFR2AC” in the “NFR Category” view directory (See Figure 4.18). In order to open the tool, this view item is selected and the “Ok” button is pressed.

As mentioned before NFR2AC tool has six tab pages, each one dealing with dedicated tasks, and moreover the first tab page “Main” is initially welcomes the user right after the tool is opened (See Figure 4.19). This tab page requests the user to select the domain and project that the current running session will be executed. User can enter and add new domains and projects with “Add” buttons. Also any selected domain and project can be removed from database with “Delete” buttons if user confirms the deletion process.

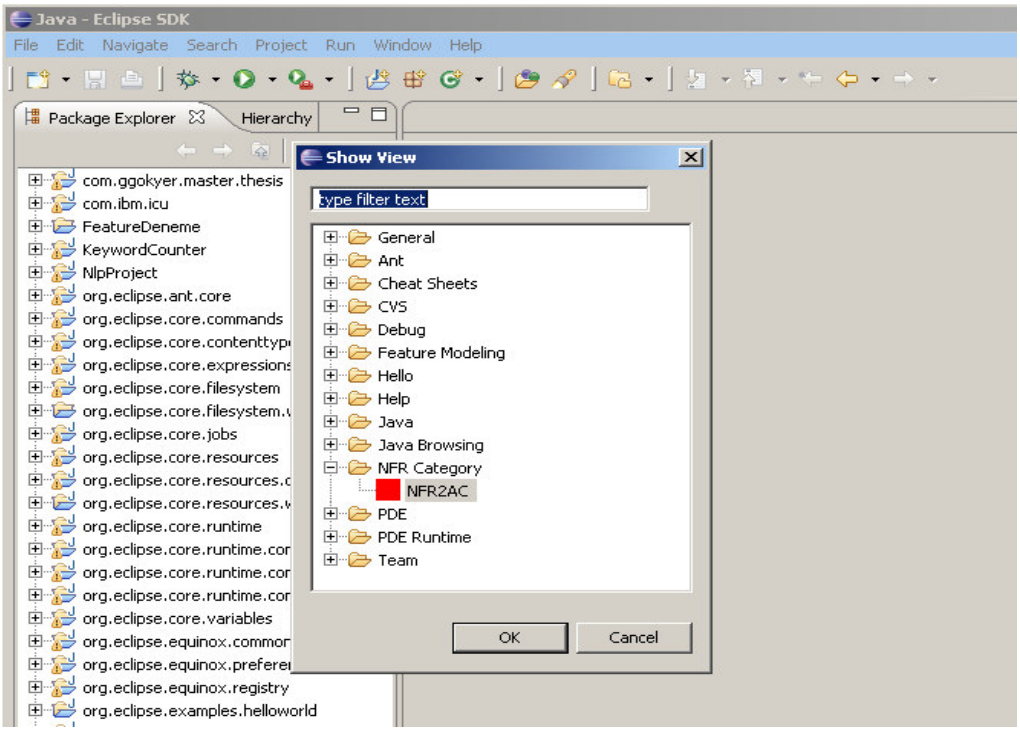

Figure 4.18 - View Item added for NFR2AC tool

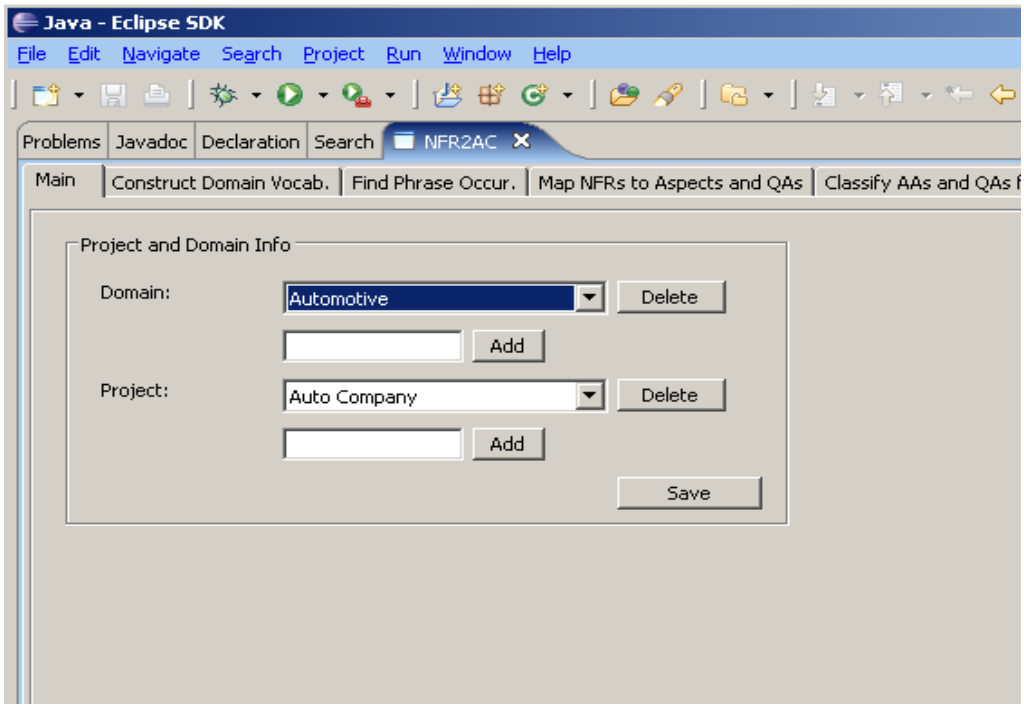

Figure 4.19 - Select domain and project

After the domain and project selection is finished "Save" button is pressed to continue the process and the next tab page becomes selected. Domain and project selection is mandatory for running the other actions of the tool, because all actions are done due to the selected domain and project. In a violation of this rule, tool reopens this tab page and forces user to make a selection.

The second tab page is designed for constructing the domain vocabulary (See Figure 4.20). User selects and loads a plain text requirement file from a file picker. When the "Load" button is pressed, the content of the file is inserted into the text area below the button. After that, user presses the "Advise" button and the key phrases within the loaded text are filtered and listed in the table in a descending order by frequency.

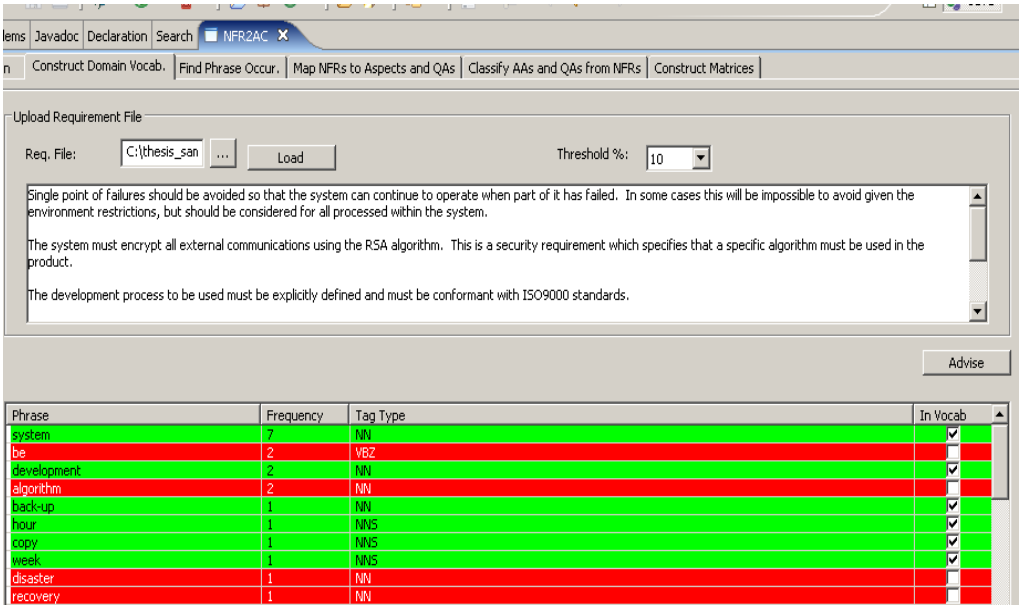

Figure 4.20 - Construct Domain Vocabulary

User also inputs a threshold value before the filtering process. This threshold information provides initial selection of the listed phrases due to the threshold probability. Let's assume that threshold is selected as "70%". Then, after the filtering process, 70 percent of phrases in the table become green and ready for adding to the domain vocabulary with selected "In Vocab" check boxes. Remaining 30 percent of phrases are red painted and initially shown out of domain vocabulary. However, user can select / deselect each phrase after the initial listing process. When the table takes its final state, user presses the "Save" button and adds the selected phrases into the domain vocabulary.

During the phrase selection process, user can also consider the cross-domain occurrence information of each phrase with pressing "Cross Domain Occ" button (See Figure 4.21). This panel gives a detailed inventory of occurrences of that phrase in previous domains and projects. So, user can decide to add or not to add any phrase after considering this information.

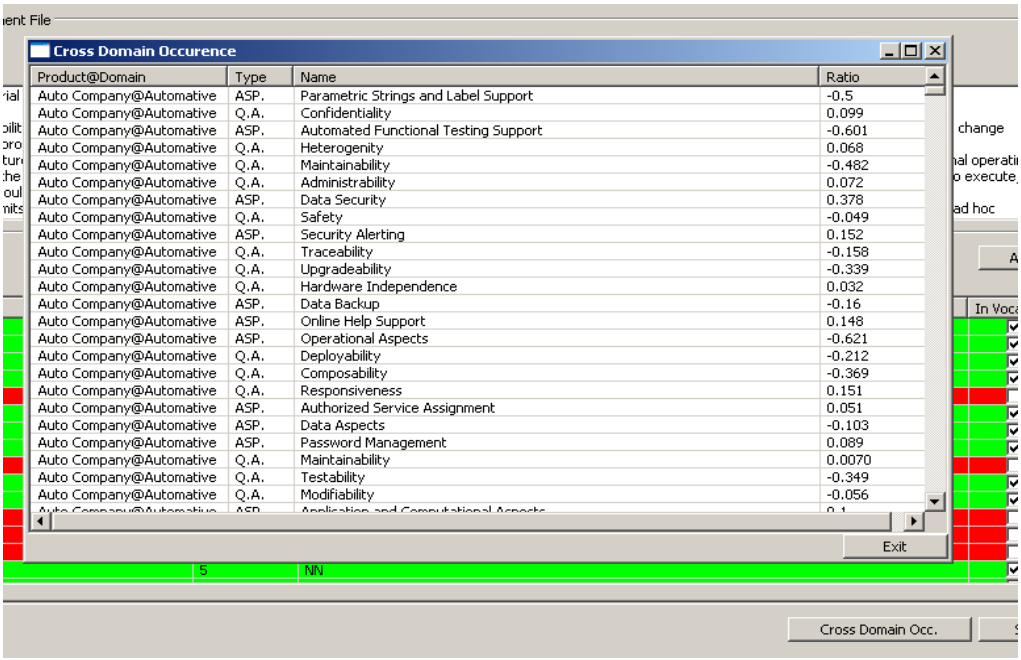

Figure 4.21 - Cross-domain occurrence for each phrase

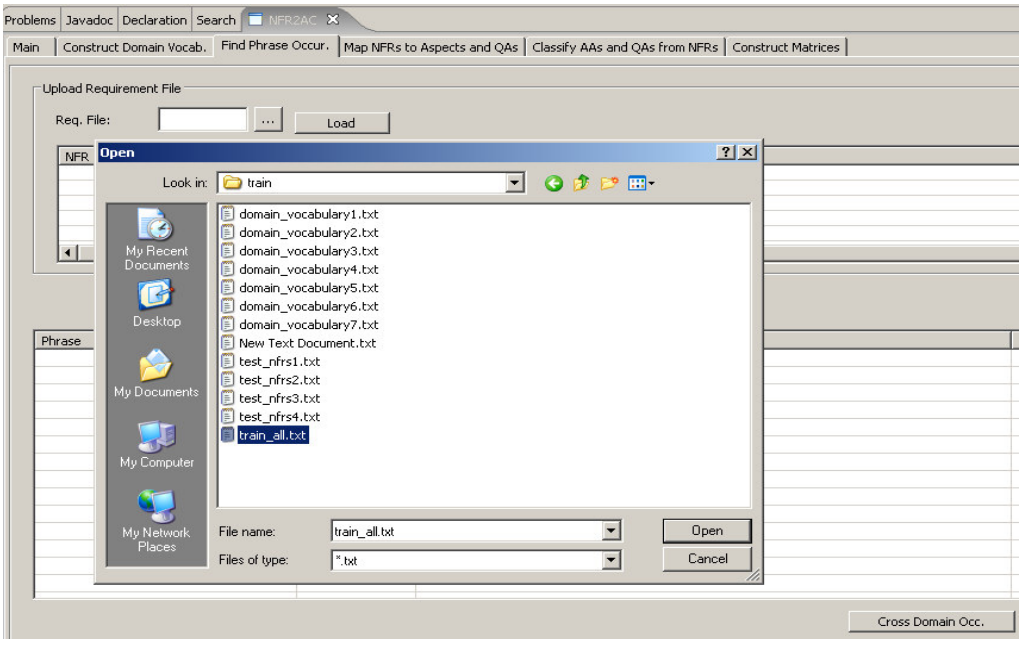

Figure 4.22 - Load NFR plain text for training

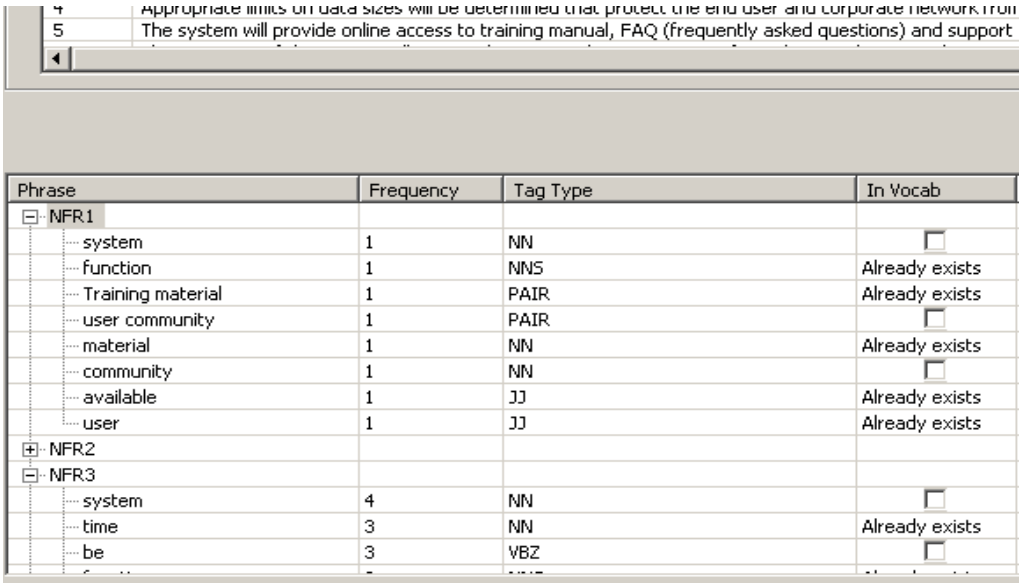

Figure 4.23 - Find key phrases for each NFR

When an acceptable enough domain vocabulary construction is completed, user can start training sessions by using this vocabulary. From the "Find Phrase Occ" tab page, user loads requirement plain texts to the system, with help of a file picker and a "Load" button like in the previous tab page (See Figure 4.22). Then NFR sentences are parsed from the plain text due to the new line character, and each NFR sentence is added to the upper table as a single row including the key phrases (See Figure 4.23).

After that, user presses the "Advise" button and for each NFR sentence, key phrases are filtered and listed in the below table with the occurrence information for each one in the current domain vocabulary. As in the previous tab page, user can reach the cross-domain occurrence information from this tab page too. When the "Save" button is pressed, domain vocabulary phrases are moved into the next tab page with their frequency information (See Figure 4.24).

In the tab page “Map NFRs to Aspects and QAs”, domain vocabulary phrases and frequencies for each NFR sentence are initially loaded to the upper table.

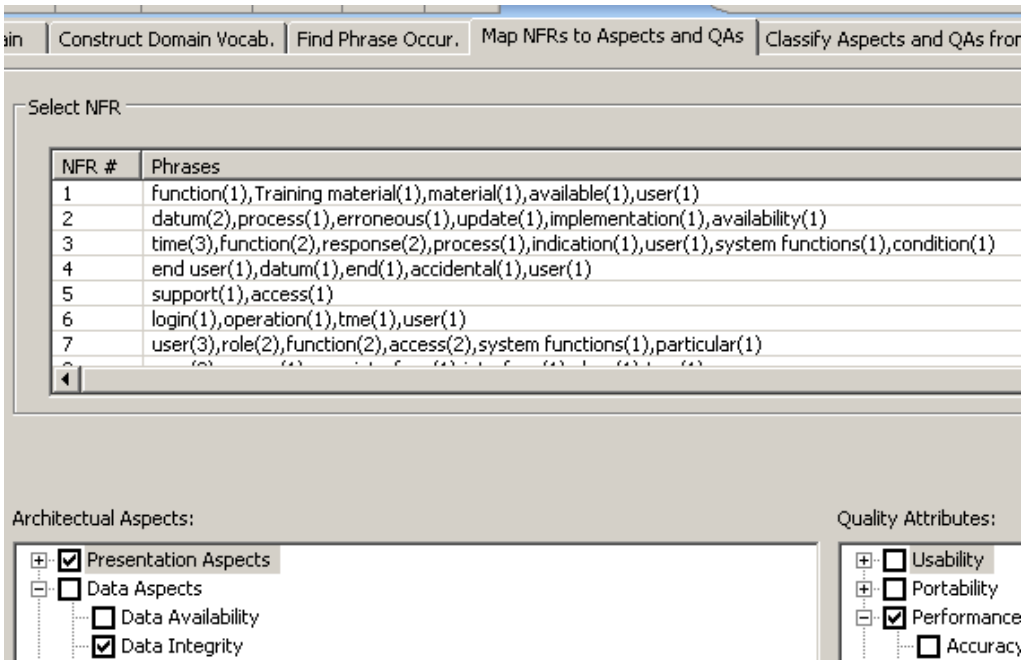

Figure 4.24 - Filter key phrases and frequencies for each NFR

When user double clicks the upper table, current architectural aspects and quality attributes are loaded to the two lists below for selected NFR sentence. Then, user marks the AAs and QAs from lists that are related to the selected NFR, finally “Train” button is pressed and mapping data is trained by the system (See Figure 4.25).

When enough training session is completed, user can start testing sessions by using constructed knowledge base. In the tab page “Classify Aspects and QAs”, user loads a plain text to the system, which AAs and QAs will be searched within (See Figure 4.26). Then, NFR sentences are parsed from the plain text due to the newline

character, and each NFR sentence is added to the upper table as a single row. After that, user presses the “Deduce” button and for each NFR sentence, occurrence probability for each architectural concern is found by SVM testing and these probabilities are listed in the below table for each NFR.

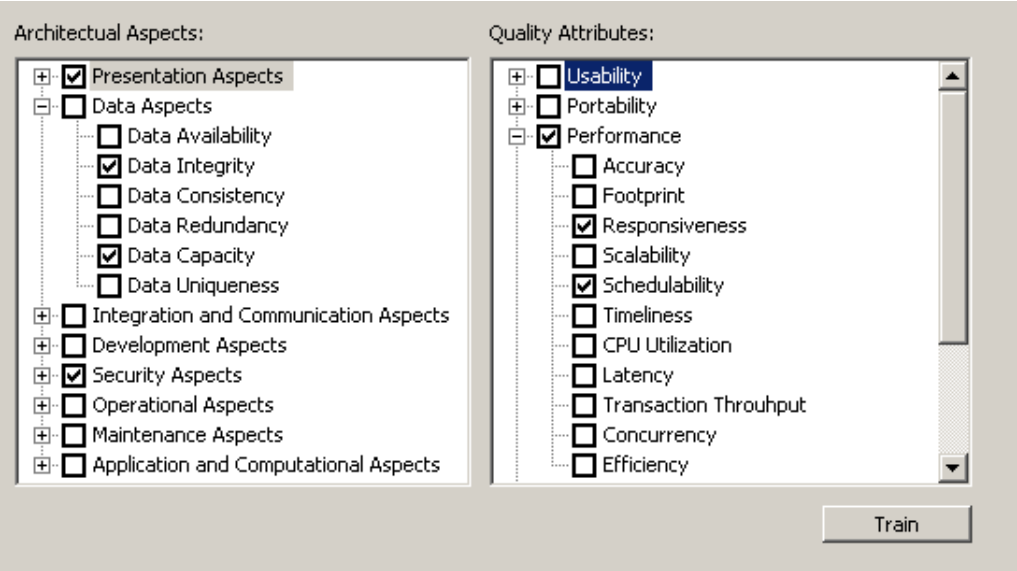

Figure 4.25 - Map Architectural Aspects and Quality Attributes for each NFR

When user double clicks the upper table, below table is updated with the testing results of the selected NFR (See Figure 4.27). If the user presses the “Save” button, a file dialog is opened and testing results are exported to an XML file (See Figure 4.28).

When enough testing session is completed, user can construct the matrices introduced in the original paper [5] in order to construct the Utility Concern Spaces (UCSs) by using constructed knowledge base (See Figure 4.29).

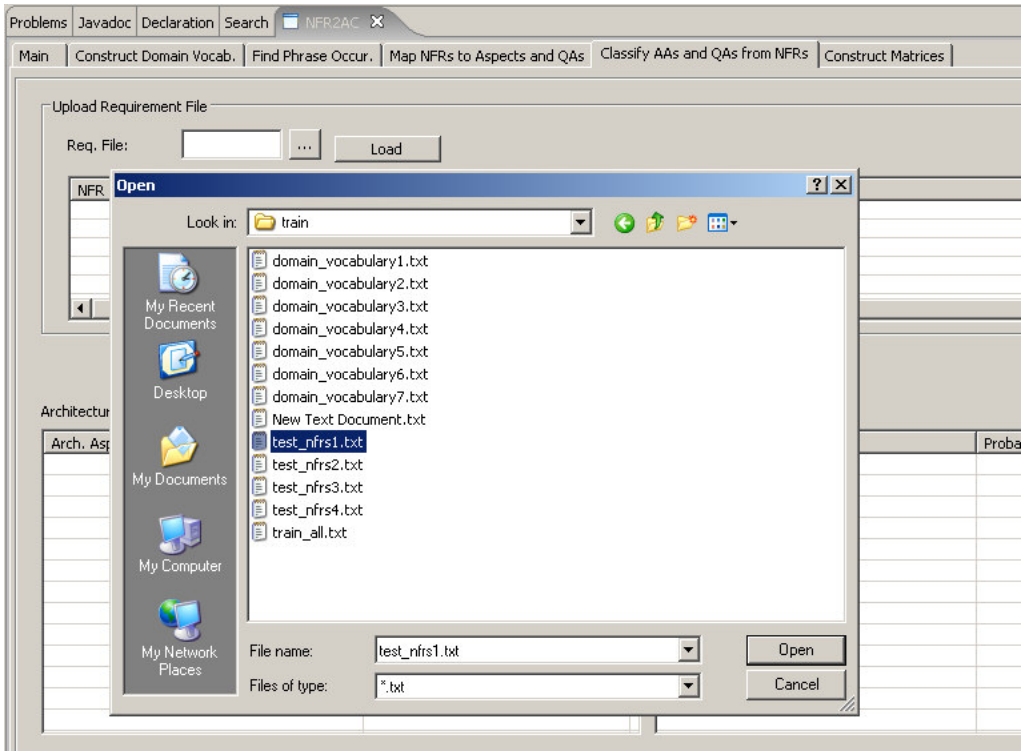

Figure 4.26 - Load NFR plain text for testing

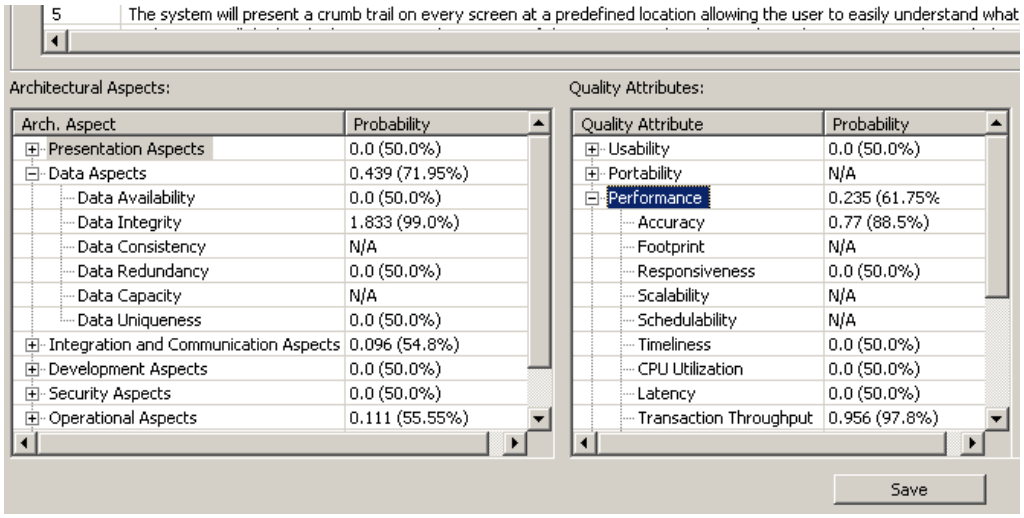

Figure 4.27 - Testing results for each NFR

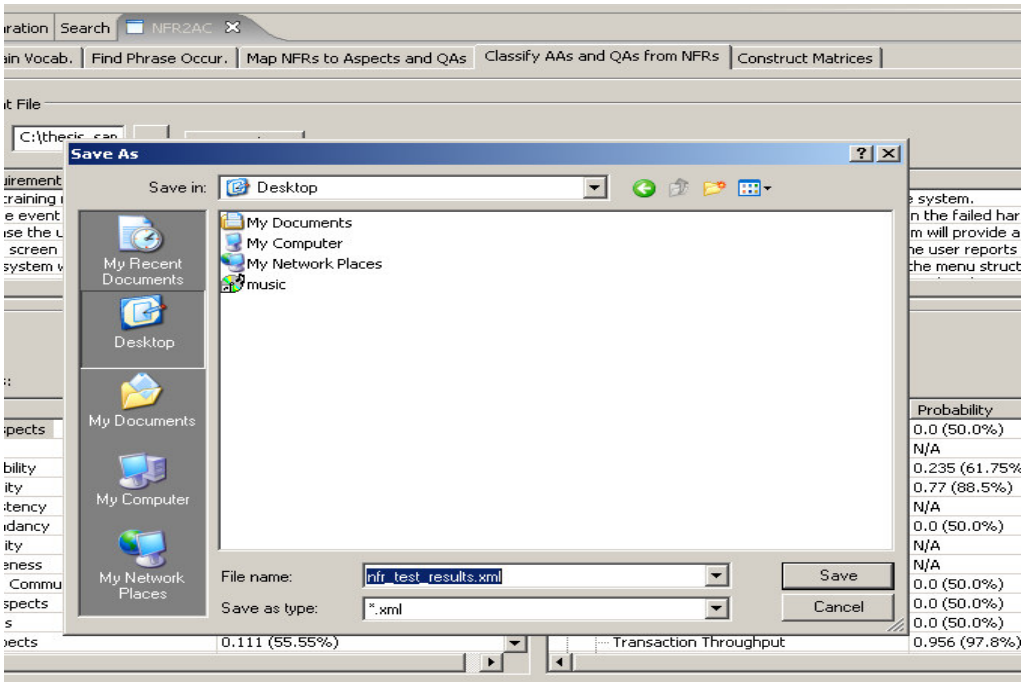

Figure 4.28 - Export testing results to an XML file

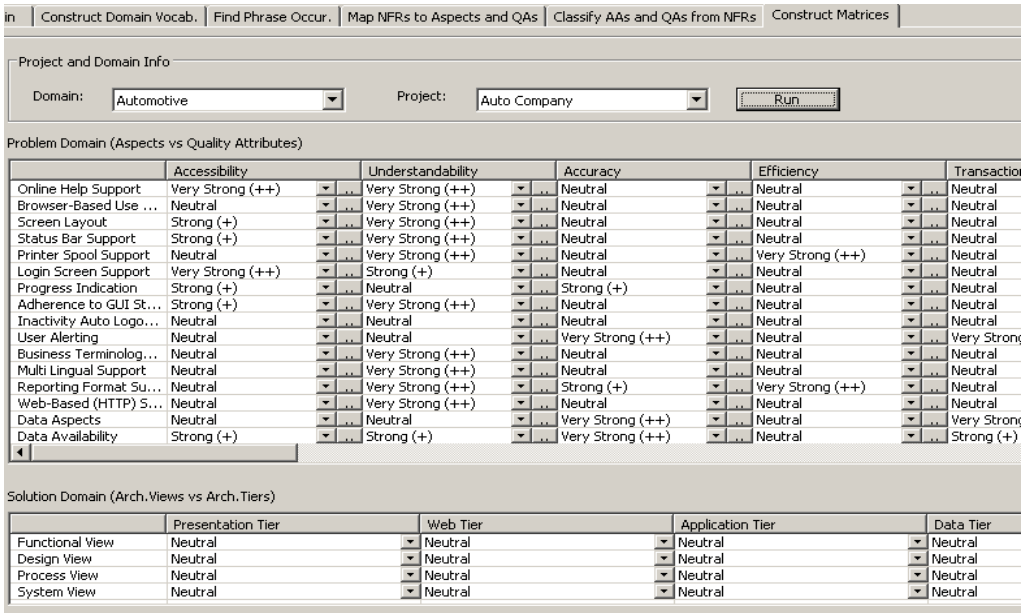

Figure 4.29 - Construct matrices for Utility Concern Spaces generation

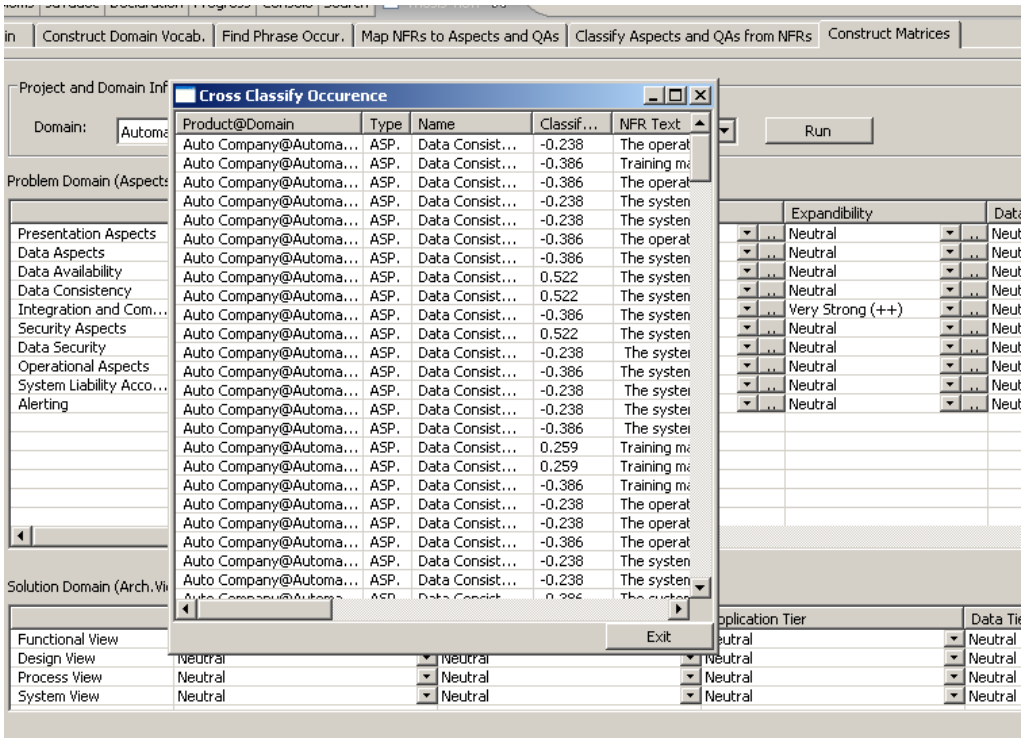

Figure 4.30 - Cross classify occurrence for each cell

In the tab page “Construct Matrices”, user selects the domain and project information. For domain and project combos, “ALL” items can be selected too in order to construct matrices due to the domains or projects globally. After the domain-project selection, user presses the “Run” button and for the selected pair, problem domain matrix is dynamically constructed including the architectural aspects and quality attributes encountered in related testing sessions due to the criteria explained in the “Proposed Approach”. Since solution domain is out of scope for this study, solution domain matrix is initially filled with default “Neutral” cell values.

User can press the small buttons in the cells of problem domain matrix in order to view the cross classify occurrences of architectural concerns of the selected cell in

previous domains and projects (See Figure 4.30). After the matrices are taken their final state by user, if the “Save” button is pressed, a file dialog is opened and final matrices are exported to an XML file (See Figure 4.31).

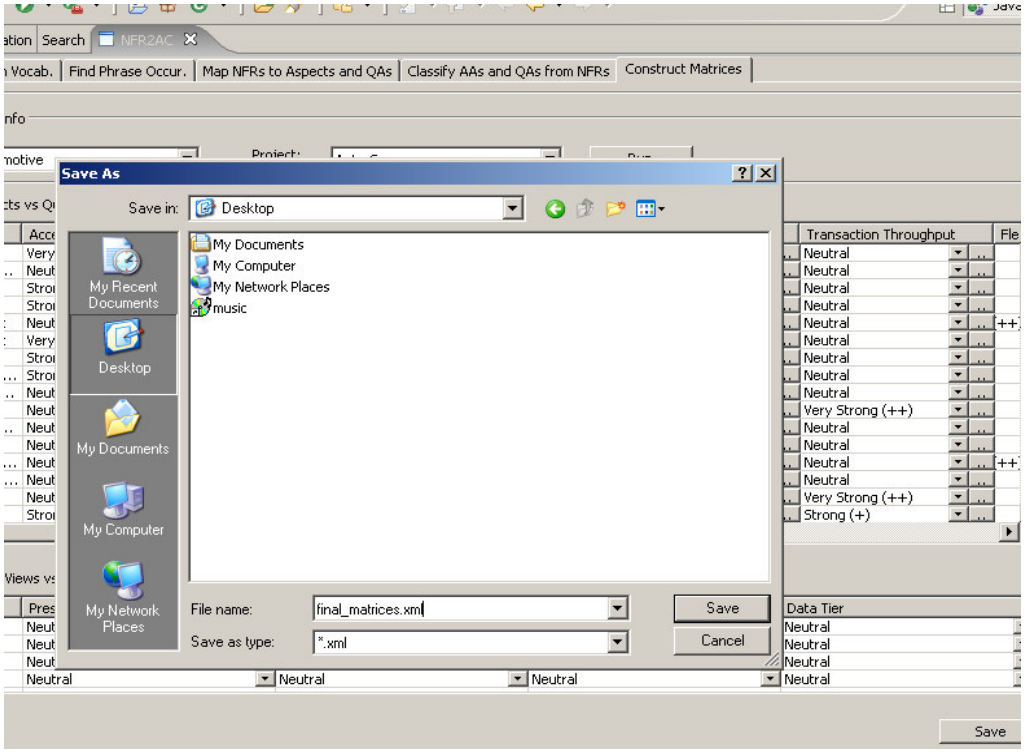

Figure 4.31 - Export matrices to an XML file

4.2 Experimentation

In order to validate the proposed approach, both of the approach and supporting toolset have been tested with a real-life project in automotive domain. Several NFRs are mapped with related ACs. Then, two main experiments have been realized. First, for the NFR set used in the testing session, the expected architectural concerns and the found architectural concerns are compared for each NFR. Then,

the success rate is calculated for this approach. Second, UCS matrix is constructed in two ways individually; by using the NFR2AC and not using it. Finally, two individual UCS matrices are compared for a success analysis. In this section, details of these processes are given step by step.

There are two NFR sets given in Appendix C2 and C3, where C2 is the list of NFRs used in training session of experimentation, and C3 is the list of NFRs used in testing/classifying session of experimentation. These NFR sets also exist under related directories in Appendix C1, with full experimentation results.

As the first step of the experimentation phase, a comparison is made between the expected architectural concerns and the found architectural concerns for each NFR used in the testing session given in Appendix C3. In this table, expected architectural concerns are given for each NFR and the bold and the italic ones are the concerns found by the SVM. The Bold concerns have bigger occurrence results than the italic ones. For getting the success rate, a set of formulas is used as follows:

$$\text{accuracy} = \frac{\text{number of true positives} + \text{number of true negatives}}{\text{numbers of true positives} + \text{false positives} + \text{false negatives} + \text{true negatives}} \tag{7}$$

$$\text{Precision} = \frac{tp}{tp + fp} \tag{8}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{9}$$

$$\text{false positive rate} = \frac{\text{number of false positives}}{\text{total number of negative instances}} \tag{10}$$

$$\text{false negative rate} = \frac{\text{number of false negatives}}{\text{total number of positive instances}} \tag{11}$$

Therefore, number of true positives, true negatives, false positives and false negatives are counted for each NFR used in testing session (See Table 4.1).

Table 4.1 – Numbers for Accuracy Analysis of NFRs used in testing session

NFR #	TP	FP	TN	FN	NFR #	TP	FP	TN	FN
1	10	1	129	0	**14**	7	13	120	0
2	10	12	117	1	**15**	5	19	116	0
3	7	13	119	1	**16**	4	2	131	3
4	5	5	128	2	**17**	6	8	126	0
5	7	1	132	0	**18**	7	9	124	0
6	9	13	117	1	**19**	4	4	131	1
7	6	30	102	2	**20**	7	11	121	1
8	7	18	115	0	**21**	6	16	115	3
9	12	20	107	1	**22**	3	8	125	4
10	6	11	123	0	**23**	3	5	130	2
11	7	16	117	0	**24**	4	9	125	2
12	4	19	116	1	**25**	9	25	105	1
13	0	0	134	6	**Total**	155	288	3027	30

According to the numbers in Table 4.1 and the formula (7), accuracy of this approach becomes 91%. For the formulas (8) and (9), precision becomes 35% and recall becomes 84% for this approach. By using the formulas (10) and (11), false positive rate becomes 9% and false negative rate becomes 7%.

Then, for the UCS matrix comparison experiment, UCS matrix is generated manually and not using the implemented toolset. Since the mappings and the matrix generation are done under the expert control, there is no any training session for this technique. Only a classification process is done for the input NFRs, and the expert finds the related ACs manually in order to generate the matrix. For the NFRs given

in Appendix C3, encountered architectural concerns are also given for each NFR in the same table.

Due to the architectural concerns encountered, expert creates the UCS matrix, and then fills the each cell of the matrix with related correlation values. Finally, the UCS matrix is manually generated and a small snapshot of UCS matrix is in Table 4.2.

Table 4.2 – UCS matrix (Manual)

	Accessibility	Understandability	Accuracy	Efficiency
Online Help Support	Neutral	Very Strong (++)	Neutral	Neutral
Browser-Based Use Support	Very Strong (++)	Neutral	Neutral	Very Strong (++)
Screen Layout	Neutral	Very Strong (++)	Neutral	Neutral
Status Bar Support	Neutral	Strong (+)	Neutral	Neutral

After getting the final UCS matrix manually, then the same process is repeated with using the implemented toolset. Initially, domain vocabulary is created with NFRs given in Appendix C2, where the phrases marked as bold and italic are added to the domain vocabulary. After that, a knowledge base is constructed by training the NFR-AC mappings again according to the NFR texts and matching architectural concerns given in Appendix C2. Finally, testing sessions are completed by loading the NFRs in Appendix C3, which are also used by the manual fillings.

The NFR2AC toolset results the final UCS matrix automatically, which is given as an XML export in related directory of Appendix C1. Again, a small snapshot of the final matrix is given at the end of the process as in Table 4.3.

Table 4.3 – UCS matrix (NFR2AC)

	Accessibility	Understandability	Accuracy	Efficiency
Online Help Support	Very Strong (++)	Very Strong (++)	Strong (+)	Neutral
Browser-Based Use Support	Neutral	Very Strong (++)	Neutral	Neutral
Screen Layout	Strong (+)	Very Strong (++)	Neutral	Neutral
Status Bar Support	Strong (+)	Very Strong (++)	Neutral	Neutral

In order to find the accuracy of the UCS matrix generation, again the accuracy formula given in (7) is used. According to the numbers in Table 4.4 and the formula (7), accuracy of this approach becomes 57%. For the formulas (8) and (9), precision becomes 59% and recall becomes 45% for this approach. By using the formulas (10) and (11), false positive rate becomes 25% and false negative rate becomes 74%.

Table 4.4 – Numbers for Accuracy Analysis of UCS matrix (NFR2AC)

TP	FP	TN	FN
197	137	302	246

4.3 Remarks

In this chapter, NFR2AC toolset is described and the implementation details have been given. Used technologies and libraries have been defined shortly. Then, NFR2AC toolset was introduced in a general way. After that, an overview of the packages and package details were given. UML diagrams and database diagrams have been drawn in order to picture the system implementation deeply. At the end of the implementation section, a user's guide was appended for explaining the usage of the toolset.

After the implementation details, with the aid of an implemented toolset, the experimentation details are given, which are done in order to approve the sense of the idea. Several NFRs are mapped with related ACs. Then, the accuracy of the system is calculated for finding the expected architectural concerns. As the next step, UCS matrices are compared. UCS matrix is constructed by using the NFR2AC and same matrix is constructed by not using the system for the same NFRs. Finally, constructed two individual UCS matrices are compared and the accuracy is given.

CHAPTER 5

CONCLUSION

5.1 Summary

This thesis proposed an approach to automatically extract architectural aspects and quality attributes from non-functional requirements expressed in plain text. In order to reach this goal, the approach utilizes NLP and ML techniques extensively for setting up an automated environment. For the NLP techniques part-of-speech tagging, phrase chunking and stemming are used. For the ML techniques SVM is used. Using the NFR2AC toolset, architects construct a domain vocabulary by filtering and adding key phrases from NFR plain texts. Later on, they map NFRs to architectural aspects and quality attributes, and train the knowledge base with this information. Finally, they ask the NFR2AC tool to deduce the occurrence probabilities of each architectural aspect and quality attribute for every NFR requested.

At the beginning, the background information has been given about Requirements Engineering, Non-Functional Requirements, Architectural Modeling, Natural Language Processing and Machine Learning. Then, the proposed approach was introduced. After that, implementation details were given and NFR2AC toolset was explained in detail. Each screen and its functionality were described one by one. After the implementation details, experimentation section has taken place. In this section, training and testing steps realized by using the system have been explained. Then, manual filling by the domain expert and the automatic filling by the system have been carried out. Finally, results of the two techniques were compared to measure the effectiveness of the proposed approach and accompanied toolset.

5.2 Conclusions

The approach suggests the idea of using a Machine Learning (ML) method based on Support Vector Machines to relate the NFRs to classified architectural aspects and quality attributes in an automated manner with the aid of a knowledge base refined by an expert. This idea created the better results when intelligent knowledge bases have been formed by means of successive trainings.

In order to empower the architects, NFR2AC toolset has been implemented as an Eclipse plug-in. Both of the approach and supporting toolset have been tested with a real-life project in automotive domain. Although the knowledge base has been originally trained with several applications from diverse business domains, the work intentionally did not take any previous case from the automotive domain in order to see the effectiveness of NFR2AC approach with a brand new business domain. The implementation of the approach by means of the toolset given in this work has revealed some practical results. Experimentation phase has been done for two steps. Main goal of this thesis was getting successful results for the first step; second step was just a trial for showing the way of the matrix generation. First step ended up with successful results, which validates the idea of this approach with enough success rates. Moreover, second step ended up with the results that around 60% of the cells for the generated UCS matrix are exactly the same with that UCS matrix expected by an individual expert system architect. It is also the fact that the domain expert has extensive knowledge on architectural modeling of transactional software systems but not deep know-how in automotive domain. This was also the intention of the validation approach to see the real effectiveness of the approach for an unknown business domain. Furthermore, it must be mentioned that manual filling of UCS matrix is a subjective process. It depends on the comment of domain expert and matrix differs from expert to expert. Therefore, success rate for the second step is not a strict result.

In spite of the acceptable success rates, there are still some steps to take for the improvement of the system. In the implementation phase, JNI Kernel Extension for

SVM^{light} software was used for the integration of Java and SVM^{light}. However, there are some bugs encountered in this software. For example, under certain conditions some exceptions occurred and SVM models had to be trained with the missing parameters. Thus, it has been concluded that a full system train without any interface bug might have ended up with better testing results.

Another deficiency might stem from the fact of using a default linear kernel for the SVM. Any other kernels and also any other ML techniques can provide more efficient models probably. Also, it has been witnessed that more and more NFRs are needed for the system to create better knowledge bases. Additionally, domain expert(s) must pay attention for the key phrase selection. Likewise, non-redundant phrases should be selected as much as possible.

Last but not least, various domain experts may construct a knowledge base for a single domain distinctly to improve the aptitude of the knowledge base during the training session, which is supposed to give much better testing results in that case. This has also revealed that every domain expert may select different NFR – architectural concern mappings during the training phase. Hence, the effectiveness of the proposed approach does not only depend on the toolset implementation but also depends on the variety and the knowledge level of the domain expert(s).

In spite of all these factors, this study has successfully put an effective roadmap for the automated architectural modeling approach that starts from NFRs specified in plain text. The approach has also been validated through the implementation of an accompanied toolset and an experimentation of a real-life project as well. By improving the knowledge base with more training, it is expected to end up with much better results that are very close to the manual fillings of domain experts.

5.3 Future work

The NFR2AC approach has been based on NLP and ML techniques where default execution methodologies and parameters are used for both of the technologies in the

implementation of the approach. Regarding the extension of this study, the work can be extended to improve the extractions from NFRs expressed in plain text. To this end, different parsers can be used for NLP, and the outcomes of NLP can be provided to ML as different SVM features instead of frequencies. Trying the ML techniques other than SVM may also further enhance the approach.

Another extension of the study can be the incorporation of NFR2AC toolset into the software product line engineering infrastructures. Also to further validate the idea, a similar approach can be worked for the solution domain in addition to study the approach for the problem domain.

Another issue might be separating the screens and the methods of the toolset into three different applications for each role ‘Business Domain Expert’, ‘Architectural Domain Expert’ and ‘Application Architect’. Also for the NLP part, speech recognition techniques can be added to the current approach.

REFERENCES

[1] Gokyer, G., Cetin, S., Sener C. and Yondem, M. T., “Non-Functional Requirements to Architectural Concerns: ML and NLP at Crossroads”, ICSEA 2008, IEEE CSP, 2008, Sliema-Malta.

[2] Baniassad, E. L. A. and Clarke, S., “Finding aspects in requirements with Theme/Doc”, Early Aspects Workshop, International Conference on AOSD, 2004.

[3] Loughran, N., Sampaio, A. and Rashid, A., "From Requirements Documents to Feature Models for Aspect Oriented Product Line Implementation", MoDELS 2005 Workshops, LNCS 3844, pp. 262-271, 2005.

[4] Rosenhainer, L., "The DISCERN Method: Dealing Separately with Crosscutting Concerns", Early Aspects Workshop at OOPSLA-2005, 2005.

[5] Cetin, S., Altintas, N. I. and Sener C., “An Architectural Modeling Approach with Symmetric Alignment of Multiple Concern Spaces”, ICSEA 2006, IEEE CSP, 2006, Tahiti – French Polynesia.

[6] Liu D., Mei H., “Mapping requirements to software architecture by feature-orientation”, STRAW'03 Second International Software Requirements to Architectures Workshop, Portland, Oregen, 2003.

[7] Robertson S., Robertson J., “Mastering the requirements process”, ACM Press, 1999, ISBN: 0 201 36046 2

[8] Franch X., Botella P., “Putting Non-Functional Requirements into Software Architecture”, Software Specification and Design, 1998, Proceedings. Ninth International Workshop, 1998.

[9] Cysneiros L.M., Leite J.C.S.P, “Using UML to Reflect Non-Functional Requirements”, Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, 2001.

[10] Yusop N., Zowghi D., Lowe D., “The Impacts of Non-Functional Requirements In Web System Projects”, Proceedings of EMCIS, 2006.

[11] Kazman R., Bass L., Abowd G. and Webb M., “SAAM: A Method for Analyzing the Properties of Software Architectures”, Proceedings of ICSE 16, pp. 81-90.

[12] Chowdhury G.G., "Natural Language Processing", Annual Review of Information Science and Technology, Vol.37, No. 1, 2003.

[13] Part-of-Speech Tutorial, http://alias-i.com/lingpipe/demos/tutorial/posTags/read-me.html, Last Access Date: 24/07/2008.

[14] Smith L., Rindflesch T., Wilbur W.J., "MedPost: a part-of-speech tagger for bioMedical text", 2004

[15] Singh B.S., "Search Algorithms", DRTC Workshops, 2003

[16] Gimenez J, Marquez L., "SVM Tool: A general POS tagger generator based on Support Vector Machines", 2004

[17] Nakagawa T., Kudoh T., Matsumoto Y., "Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines", In Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium, 2001.

[18] Support Vector Machines, http://www.neural-forecasting.com/support_vector_machines.htm, Last Access Date: 24/07/2008.

[19] Joachims T., "Text Categorization with Support Vector Machines: Learning with Many Relevant Features", Proceedings of ECML, Springer, 1998.

[20] Cleland-Huang, J., Settimi, R., Zou, X. and Solc, P., "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects", Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), 2006.

[21] Steele A., Arnold J. and Cleland-Huang J., "Speech Detection of Stakeholders' Non-Functional Requirements", First International Workshop on Multimedia Requirements Engineering (MERE'06 - RE'06 Workshop), p. 3, 2006.

[22] Hall, J. G., Jackson, M., Laney, R., Nuseibeh, B. and Rapanotti, L., "Relating Software Requirements and Architectures Using Problem Frames", Proceedings of RE'02, Essen, 2002.

[23] Baniassad, E. L. A., Clements, P., Araujo, J., Moreira, A., Rashid, A. and Tekinerdogan, B., "Discovering Early Aspects", IEEE Software. Volume 23, pp. 61-69, 2006.

[24] Chung L., Nixon B., Yu E., Mylopoulos J., "Non-Functional Requirements in Software Engineering", The Kluwer international series in software engineering, 1999, ISBN: 0-7923-8666-3.

[25] Cysneiros L.M., Leite J.C.S.P, Neto J.M.S., "A Framework for Integrating Non-Functional Requirements into Conceptual Models", Requirements Engineering Journal – Vol 6, Issue 2 Apr. 2001, pp: 97-115.

[26] Li, K., Dewar, R. G. and Pooley, R. J., "Towards Semi-automation in Requirements Elicitation: mapping natural language and object-oriented concepts", Doctoral Consortium, 13th IEEE International Conference on Requirements Engineering (RE'05), 2005.

[27] Sampaio, A., Rashid, A., Rayson, P., "Early-AIM: an approach for identifying aspects in requirements", poster presentation in Requirements Engineering, Proceedings of 13th IEEE International Conference on Requirements Engineering, 2005.

[28] Sampaio, A., Loughran, N., Rashid, A. and Rayson, P., "Mining Aspects in Requirements", Workshop on Early Aspects at AOSD-2005, 2005.

[29] Chitchyan R., Sampaio A., Rashid A. and Rayson P., "Evaluating EA-Miner: Are Early Aspect Mining Techniques Effective?", Proceedings of First International Workshop Towards Evaluation of Aspect Mining (TEAM 2006) co-located with 20th European Conference on Object-Oriented Programming (ECOOP 2006), Nantes, France, July 4, 2006.

[30] Jonsson K., Kittler J., Li Y.P., Matas J., "Support Vector Machines for Face Authentication", Proceedings of the Tenth British Machine Vision Conference, 1999.

[31] Larsson, M., "Predicting quality attributes in component-based software systems", Ph.D. Thesis, Department of Computer Science and Engineering, Malardalen University, http://www.idt.mdh.se/~icc/phd/MagnusLarsson/magnus%20phd%20FINAL-A4.pdf, 2004.

[32] Hsu, C. W., Chang, C. C., and Lin, C. J., "A practical guide to support vector classification," LIBSVM — A library for Support Vector Machines, http://www.csie.ntu.edu.tw/~cjlin/libsvm, Last Access Date: 24/07/2008.

APPENDIX A

MEDPOST PART-OF-SPEECH TAG SET

In this section, MedPost part-of-speech tag set is given.

CC	coordinating conjunction	RR	adverb	VHG	participle *having*
CS	subordinating conjunction	RRR	comparative adverb	VHI	infinitive *have*
CSN	comparative conjunction (than)	RRT	superlative adverb	VHN	participle *had*
CST	complementizer (that)	SYM	symbol	VHZ	3rd pers. sing. *has*
DB	predeterminer	TO	infinitive marker *to*	VVB	base form lexical verb
DD	determiner	VM	modal	VVD	past tense
EX	existential *there*	VBB	base *be, am, are*	VVG	present part.
GE	genitive marker *'s*	VBD	past *was, were*	VVI	infinitive lexical verb
II	preposition	VBG	participle *being*	VVN	past part.
JJ	adjective	VBI	infinitive *be*	VVZ	3rd pers. sing.
JJR	comparative adjective	VBN	participle *been*	VVNJ	prenominal past part.
JJT	superlative adjective	VBZ	3rd pers. sing. *is*	VVGJ	prenominal present part.
MC	number or numeric	VDB	base *do*	VVGN	nominal gerund
NN	noun	VDD	past *did*	(	left parenthesis
NNP	proper noun	VDG	participle *doing*	)	right parenthesis
NNS	plural noun	VDI	infinite *do*	,	comma
PN	pronoun	VDN	participle *done*	.	end-of-sentence period
PND	determiner as pronoun	VDZ	3rd pers. sing. *does*	:	dashes, colons
PNG	genitive pronoun	VHB	base *have*	“	left quote
PNR	relative pronoun	VHD	past *had*	”	right quote

Figure A.1 - MedPost part-of-speech tag

APPENDIX B

SAMPLE FILES

In this section, some sample files are given used or produced in NFR2AC toolset.

B.1 An SVM model file

```
SVM-light Version V6.01
0 # kernel type
3 # kernel parameter -d
1.0 # kernel parameter -g
1.0 # kernel parameter -s
0.0 # kernel parameter -r
empty# kernel parameter -u
94 # highest feature index
2 # number of training documents
3 # number of support vectors plus 1
0.39165004510296875 # threshold b, each following line is a SV (starting with alpha*y)
-5.367793159775216 1:0.13333334028720856 7:0.13333334028720856 9:0.13333334028720856
11:0.13333334028720856 15:0.13333334028720856 18:0.13333334028720856
23:0.03333333507180214 42:0.03333333507180214 46:0.03333333507180214
57:0.03333333507180214 93:0.03333333507180214 94:0.03333333507180214
5.367793159775216 2:0.4444444477558136 25:0.1111111119389534 55:0.1111111119389534
62:0.1111111119389534 75:0.1111111119389534 76:0.1111111119389534
```

B.2 An XML export file of NFR2AC test results

```
<?xml version="1.0" encoding="UTF-8"?>

<Thesis_ggokyer>
        <ClassifyResults>
                <DomainName>Finance</DomainName>
                <ProjectName>Bank ABC</ProjectName>
                <RunId>11b047d0c740237</RunId>
                <NfrBoundary>
                        <NfrId>1</NfrId>
                        <NfrValue>Training material will be made available to the user community
to assist them in installing and using the functions of the system. It is anticipated that under normal
use and normal operating conditions of the Information System infrastructure the system always
provides a reasonable response time. The system  GUI graphical user interface will be designed and
implemented according to the guidelines that exist within TME IS. For known complicated functions
that take a long time to execute, the system should provide indication to the user that a process is
being executed through some visual clue.</NfrValue>
                        <NfrPhraseList>
                                <NfrPhrase>
                                        <Code>1</Code>
                                        <Value>user</Value>
                                        <Frequency>3</Frequency>
                                </NfrPhrase>
                                <NfrPhrase>
```

```
                                        <Code>3</Code>
                                        <Value>function</Value>
                                        <Frequency>2</Frequency>
                                </NfrPhrase>
                        </NfrPhraseList>
                        <AspectList>
                                <Aspect>
                                        <Id>1</Id>
                                        <Name>Presentation Aspects</Name>
                                        <Ratio>0.117</Ratio>
                                </Aspect>
                                <Aspect>
                                        <Id>1.1</Id>
                                        <Name>Online Help Support</Name>
                                        <Ratio>0.148</Ratio>
                                </Aspect>
                        </AspectList>
                        <QualityAttributeList>
                                <QualityAttribute>
                                        <Id>1</Id>
                                        <Name>Usability</Name>
                                        <Ratio>-0.084</Ratio>
                                </QualityAttribute>
                                <QualityAttribute>
                                        <Id>1.1</Id>
                                        <Name>Accessibility</Name>
                                        <Ratio>-0.177</Ratio>
                                </QualityAttribute>
                        </QualityAttributeList>
                </NfrBoundary>
        </ClassifyResults>
</Thesis_ggokyer>
```

B.3 An XML export file of NFR2AC matrices

```
<?xml version="1.0" encoding="UTF-8"?>

<Thesis_ggokyer>
        <ClassifyResults>
                <DomainName>Automative</DomainName>
                <ProjectName>Auto Company</ProjectName>
                <ProblemDomain>
                        <Cell>
                                <Aspect>Presentation Aspects</Aspect>
                                <QualityAttribute>Portability</QualityAttribute>
                                <Value>Strong (+)</Value>
                        </Cell>
                        <Cell>
                                <Aspect>Presentation Aspects</Aspect>
                                <QualityAttribute>Hardware Independence</QualityAttribute>
                                <Value>Neutral</Value>
                        </Cell>
                </ProblemDomain>
                <SolutionDomain>
                        <Cell>
                                <View>Functional View</View>
                                <Tier>Presentation Tier</Tier>
                                <Value>Neutral</Value>
```

```
</Cell>
<Cell>
        <View>Functional View</View>
        <Tier>Web Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Functional View</View>
        <Tier>Application Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Functional View</View>
        <Tier>Data Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Design View</View>
        <Tier>Presentation Tier</Tier>
        <Value>Very Strong (++)</Value>
</Cell>
<Cell>
        <View>Design View</View>
        <Tier>Web Tier</Tier>
        <Value>Strong (-)</Value>
</Cell>
<Cell>
        <View>Design View</View>
        <Tier>Application Tier</Tier>
        <Value>Very Strong (++)</Value>
</Cell>
<Cell>
        <View>Design View</View>
        <Tier>Data Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Process View</View>
        <Tier>Presentation Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Process View</View>
        <Tier>Web Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Process View</View>
        <Tier>Application Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>Process View</View>
        <Tier>Data Tier</Tier>
        <Value>Neutral</Value>
</Cell>
<Cell>
        <View>System View</View>
        <Tier>Presentation Tier</Tier>
        <Value>Neutral</Value>
```

```
                        </Cell>
                        <Cell>
                                <View>System View</View>
                                <Tier>Web Tier</Tier>
                                <Value>Strong (-)</Value>
                        </Cell>
                        <Cell>
                                <View>System View</View>
                                <Tier>Application Tier</Tier>
                                <Value>Strong (-)</Value>
                        </Cell>
                        <Cell>
                                <View>System View</View>
                                <Tier>Data Tier</Tier>
                                <Value>Neutral</Value>
                        </Cell>
                </SolutionDomain>
        </ClassifyResults>
</Thesis_ggokyer>
```

APPENDIX C

SOURCE CODE AND EXPERIMENTATION

In this section, content of the Thesis CD is listed and NFR sets are given which are used in experimentation phase.

C.1 Thesis CD Content

- NFR2AC
  - source
  - javadoc
  - plug-in
  - model directory
- Experimentation
  - Training files
  - Testing files
  - Results

C.2 NFRs used in Training Session

Table C2.1 - NFRs used in Training Session

Req. #	DESCRIPTION	Architectural Aspects*	Quality Attributes
TR1	A ***"System Liability Agreement"*** will be prepared to accompany the project implementation that defines the availability of the system, updates of erroneous data, and ***change management processes*** for data and / or ***reports***.	6.2	4.1, 4.3, 4.4, 4.6, 7.2
TR2	Due to the nature of the used technology, no specific ***response time*** commitments will be made for system functions. It is anticipated that under normal use and normal operating conditions of the Information System infrastructure the system always provides a reasonable ***response time***. For known complicated functions that take a ***long time*** to ***execute***, the system should provide least ***latency*** and also indication to the user that a process is being executed through some visual clue.	1.17, 8.2, 8.3	3.3, 3.6, 3.7, 3.8
TR3	Appropriate ***limits*** on ***data sizes*** will be determined that protect the ***end user*** and corporate ***network*** from ***accidental requests*** for excessively ***large datasets*** (e.g. an ad hoc query that is going to return 1 million rows may be deemed to be too ***large*** and will not be ***executed***).	3.1	1.5, 3.8, 3.11
TR4	The system will provide ***online access*** to ***training manual***, ***FAQ*** (frequently asked questions) and support information.	1.1, 6.3, 6.6, 6.7	1.1, 1.3, 1.5, 1.6

* Architectural Aspects and Quality Attributes in the table are numbered according to the taxonomies given in Table 3.1 and Table 3.2.

Table C2.1 cont'd

TR5	The operation of the system will require the users to have a TME mainframe ***login*** and ***password***. There can't be any ***access*** to the system without ***permission***	5.6, 6.4, 6.5, 1.15	7.3, 8.3, 8.4, 8.5
TR6	***Access*** to particular functions in the system will be restricted depending on the ***role*** that has been allocated to ***authorized*** users. The system should restrict access to all system functions to users that have been specifically given ***authentication***. Where required, levels of ***authority*** should be granted and respected such that certain users may carry out restricted roles.	5.5, 6.5	1.1, 7.2, 8.5
TR7	The system's ***GUI*** (***graphical user interface***) will be designed and implemented according to the guidelines that exist within TME IS. The system should be intuitive to the user. This involves using clear and simple ***screen design*** with well-named ***navigation*** and action items such as ***menus*** and ***buttons***.	1.2, 1.11, 1.12	1.3, 1.5, 1.6
TR8	The system should respect the ***security*** of other systems ***security requirements*** and in doing so not breach any existing ***security procedures***.	5.7, 3.2	5.3, 5.6, 8.4, 8.5
TR9	The system should ensure that any accidental exposure to ***security risks*** (such as user leaving system ***authenticated*** whilst away from desk) should be minimized by employing normal practices such as ***session timeouts***.	5.8, 1.3	7.3, 8.1, 8.3, 8.5
TR10	The system will be used to support business ***hours*** (generally 6:30 – 19:30 CET) and therefore should be available 100% of the ***time*** unless prior arrangement agreed with the users. The system is not ***mission-critical*** and does not require 24/7 availability.	2.1, 8.4, 8.5	1.5, 3.1, 7.2, 3.6

Table C2.1 cont'd

TR11	The system is intended to support a key business ***activity*** and therefore every effort should be made to ensure the ***output*** is ***precise*** and ***reliable*** according to the definitions provided in the system business rules document.	8.6, 1.4	6.2, 8.1, 8.3
TR12	The system should behave in a ***consistent*** and ***repeatable*** manner (i.e. same ***input*** provides same ***output***). Where possible, the system should encourage standard procedures so that one thing can only be done one way ensuring that the system will behave in a standard way each time a function is used.	8.6, 8.7, 1.5	4.7, 8.1, 8.3
TR13	The system will be designed in such a way that a function can be ***rerun*** in the ***event*** of a ***system error*** without requiring significant special activities. This will mitigate any ***risks*** of data ***corruption*** in the event of a system error.	8.9, 1.4, 2.2, 5.8	3.1, 3.9, 5.8
TR14	The system should handle the ***volume*** of ***data*** equivalent to the normal business condition and be able to handle ***expansion*** expected for 5 ***years***. Where possible, ***redundant*** data should be ***archived*** or ***deleted***	6.1, 7.1, 2.4, 2.6, 8.4	1.5, 4.1, 4.4, 3.6
TR15	The system will use the ***terminology*** and ***nomenclature*** of the target business domain so that data ***labels***, functions etc are familiar to the users.	1.6, 4.5	1.3, 1.6, 9.1
TR16	The system will provide ***reporting*** facilities in ***formats*** that are handled by normal ***end user*** computing equipment (***Windows PC***) and applications, in particular: ***MS Word***, ***MS Excel***, ***Adobe*** PDF. This end user computing equipment and ***applications*** are however, not considered as being part of the system and will therefore not be part of any SLA related support.	8.8, 1.8, 1.14	1.3, 5.1, 5.3, 5.6

Table C2.1 cont'd

TR17	The system will be ***web based*** and will therefore be ***accessible*** to the user using a standard installed and controlled ***web browser***.	1.9, 1.10	1.6, 7.2
TR18	The system should respect the ***performance*** and requirements of other system residing in a shared environment, and not unduly cause any impact to such systems either in ***CPU usage*** or ***storage space***.	3.3, 8.1, 8.10	3.7, 3.11, 5.3
TR19	The ***user interface*** of the system will be available in the ***English language***.	1.2, 1.10, 1.7	1.3, 1.6
TR20	Single point of failures should be avoided so that the system can continue to ***operate*** when part of it has ***failed***. In some cases this will be impossible to avoid given the ***environment restrictions***, but should be considered for all processed within the system.	8.5, 8.10, 2.4, 3.4	1.5, 7.2
TR21	The system must ***encrypt*** all ***external communications*** using the ***RSA*** algorithm. This is a ***security*** requirement which specifies that a specific algorithm must be used in the product.	5.4	7.3, 8.5
TR22	The ***development process*** to be used must be explicitly defined and must be conformant with ***ISO9000 standards.***	4.1, 4.2, 4.3	9.4, 6.2, 6.4, 6.9
TR23	Management ***reports*** setting out the effort expended on each identified system component must be ***produced*** every two ***weeks***.	1.8	1.3
TR24	A ***disaster recovery plan*** for the system ***development*** must be specified.	7.1, 7.3	4.8, 8.4
TR25	All system data must be backed up every 24 ***hours*** and the ***back-up*** copies stored in a ***secure*** location in ***encrypted*** form which is not in the same building as the system.	6.1, 7.1, 5.4, 8.4, 3.4	8.4, 8.5, 3.6

Table C2.1 cont'd

TR26	The system shall not ***permit operation*** if the external temperature is below 4 degrees Celsius**.**	6.2, 6.3	8.4, 8.5
TR27	***Mailings*** for customers should be sorted into the three zip codes that lie within the city ***limits*** in order to get the reduced postal rate.	1.5, 3.1	1.5, 1.6
TR28	A ***License Certificate*** for a new Business must be ***mailed*** within two ***days*** of application approval	8.4, 8.6	3.6
TR29	It must be possible to ***trace*** each ***license***, its renewals, and tax payments over the ***last*** five tax ***years***.	8.4, 7.4	6.4, 3.6
TR30	Component *C* will be fully operational for *P %* of the ***time*** over a ***continuous*** measured *period* of 30 ***days*** (equivalent to 43 ***minutes*** downtime)	8.4	3.6
TR31	Component *C* will support a ***concurrent*** group of *U* users ***running*** pre-defined acceptance script *S* ***simultaneously***	8.4, 8.9	3.6, 3.10
TR32	Acceptance script *S* completes within *T* ***seconds*** on an unloaded system, and within *T2* seconds on a system ***running*** at ***maximum capacity***	8.4	3.6, 3.7
TR33	Component *C* will ***support*** *N1* ***transactions*** per ***second***, of type *T*, on an unloaded system, and *N2* transactions per ***second***, of type *T*, on a system loaded.	8.4, 8.1	3.6, 3.9
TR34	To ***format*** numbers, the number of significant digits to which accuracy should be maintained in all ***numerical calculations*** is 10	1.5, 1.8	1.3, 1.6

C.3 NFRs used in Testing Session

Table C3.1 - NFRs used in Testing Session

Req. #	DESCRIPTION	Architectural Aspects	Quality Attributes
TE1	The training manual and FAQ will be made available to the user community to assist them in installing and using the functions of the system.	**1.1**, **6.6**, **6.7**	**1.1**, *1.3*, *1.5*, *1.6*
TE2	In the event of a hardware error, the system should be able to be started on a different machine without requiring data existing on the failed hardware. Sufficient and timely back-up of data should be taken to ensure this process is possible.	*6.1, 7.1, 3.4, 5.4*	*8.4, 8.5*
TE3	In case the user enters input data into the system, which does not adhere to the business rules, specified for that data, the system will provide a clear indication to the user via some form of error message : .	*1.5*, 1.2, *1.4*	*3.9*, **4.7**
TE4	Each screen inside the system will be clearly identified by means of a unique number. This facilitates later problem solving in case the user reports an issue with the system.	*1.2*, 7.2	*1.3, 1.5*
TE5	The system will present a crumb trail on every screen at a predefined location allowing the user to easily understand what part of the menu structure he is navigating.	*1.2*, **1.11**	*1.3, 1.5, 1.6*
TE6	Each screen will display the login name and user name of the user currently authorized into the system together with the role allocated to the user.	*1.2*, **1.12**, *5.6*	*1.3, 8.4, 8.5*

Table C3.1 cont'd

TE7	Where the system required access the locale data (date, time, time zone) of the end user computing equipment (Windows PC) the user is responsible for making sure that the locale is correctly configured. Such conditions will be clearly identified and highlighted in the training manual.	1.13, **6.6, 6.7**	1.3, *1.5*
TE8	Any direct printing functionality of the reports that the system offers will be performed using the printers that are normally accessible through the end user computing environment. The correct setup of these devices is, however, outside the scope of the system. Report formats must be considered.	*1.8*, **1.14**	*7.2, 1.3*
TE9	When the user attempts to access an application page from within the web browser the system will validate whether or not the user is already logged in. If not, the system will present the login screen and enter the application in the normal way (as if the user had selected the login page directly). In this particular situation the system will not directly navigate to the page originally requested by the user.	*1.15, 5.6*, **1.9**, *1.10*	*1,6, 7.3, 8.4, 8.5*
TE10	If the system performs a lengthy operation that displays a result to the user and the user's session is timed out before the result page is displayed, the system will require the user to login again before displaying the result page.	**1.3**, *1.15*	*8.3, 8.5*

Table C3.1 cont'd

TE11	The system shall not use hard-coded strings and labels. Although the application must be available in English only, strings and labels should be organized in a clean properties file that eventually could be translated in another language.	**1.7, 4.5**	*1.3, 1,6*
TE12	The system shall ensure that data is protected from unauthorized access. Only authorized users can enter the system with secure authentication.	**5.5**, *5.6*	*8.5*
TE13	The System service X shall have an availability of 999/1000 or 99%. This means that out of every 1000 requests for this service, 999 must be satisfied.	2.1, 8.5	7.2
TE14	The system shall be developed for Windows PC platform. Reports must be available for MS Word, MS Excel, Adobe PDF programs. This is a portability requirement which affects the way in which the system may be designed.	**1.8, 8.8**	**5.3, 5.6**
TE15	The access permissions for system data may only be changed by the system's data administrator	**5.6**	*8.4, 8.5*
TE16	All external communications between the system's data server and clients must be encrypted	6.2, **5.4**	*8.4, 8.5*
TE17	The system shall not permit operation unless the operator guard is in place	*6.2*, **6.3**	*8.4, 8.5*
TE18	Mailings for customers must be designed to fit in a window envelope.	*1.5, 3.1*	*1.5, 1.6*
TE19	A License Certificate for a renewal must be mailed within seven days of receipt of the renewal.	*8.4*, **8.6**	3.6

Table C3.1 cont'd

TE20	It must be possible to trace the ownership of a Business over at least the last five years and at least the last two owners	*8.4*, **7.4**	**6.4**, 3.6
TE21	Multiple formats along the lines of Component *C* will provide sufficient capacity for *U* users each with *M* Mb of files. Component *C* will provide sufficient capacity for archived reports for *MM* calendar months at a report creation rate of *R* per calendar month	*8.4, 1.8*	3.6, 3.7, 1.6
TE22	All errors encountered during events *E1*, *E2*, *E3* are to be captured by component *C* and reported into the standard monitoring tool *MT*	1.8, **1.4**	**3.9**, 1.3
TE23	Component *C* will support an increase of factor *F* in the number of users *U* while running acceptance script *S* and maintaining latency within *P* % of the latency NFR	**8.2**	3.3, 3.8
TE24	The response time of the system should always be less than 5 seconds	*8.4*, **8.2**	3.3, 3.6
TE25	Experienced officers should be able to use all the system functions after a total training of two hours. After this training, the average number of errors made by experienced officers should not exceed two per day.	*8.4, 1.4*, 6.7	*1.3, 3.9*