A Framework to Separate Non-Functional Requirements for System Maintainability

MAHRUKH UMAR* AND MUHAMMAD NAEEM AHMED KHAN**

Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad, Pakistan.
E-mail: mahrukhumar@yahoo.com*, mnak2010@gmail.com**

ABSTRACT

Requirement Engineering (RE) is the vital part of software development life cycle - the core of any successful software is to understand its requirements. In software development, Non-Functional Requirements (NFRs) are usually considered as one of the difficult area to deal with. However, there is still lack of awareness for modeling and formalizing NFRs as they are often ambiguously specified by the users. This paper analyzes various techniques, frameworks and recommendations from various literatures; and identifies their limitations to perform critical analysis on various NFRs techniques. In addition, the paper intends to propose architecture to separate NFRs and FRs during the software development to improve system maintainability. Further, the paper introduces a process model within the architecture to specifically identify, realize and implement NFRs. The proposed architecture and NFR-process model is supplemented with the validation of a case study for a library system.

Keywords: Non-functional Requirements; Requirement Engineering; Functional Requirements; Decomposition of NFRs; Separation of Concerns.

INTRODUCTION

Non-Functional Requirements (NFRs) along with Functional Requirements (FRs) are the most significant and critical requirements for software development (Gross, D & Tu, 2001; Chung, L. et al., 2000). But, many IT companies merely concentrate on functional requirements and NFRs are only accounted for the purpose of documenting them in software requirement specifications (Umar, M. & Khan, 2011). However, there is no consensus to describe NFRs as various terms are used for example, properties, constraints, attributes and so forth. NFRs are usually considered as a part of FRs during software development; as a result, when software application is implemented, it increases complexity within the code and software artifacts gets cumbersome to maintain.
Without proper attention to NFRs, software may lacks in reliability, performance, usability and other attributes. However, there is still lack of awareness for modeling and formalizing NFRs as they are often ambiguously specified by the users. In addition, any specific NFR-process model is hardly available to elicit, model or implement these requirements separately from FRs to reduce software complexity. Keeping in view that FRs and NFRs are equally important for any software system, this paper provides an extensive review on NFR techniques and methods to specify, model and verify them, and focuses on the limitations of the existing NFR implementation methods to present a critical analysis of existing techniques. Furthermore, in this paper, we propose architecture to separate these requirements during software development process. Traditional process models such as SDLC, iterative and incremental development etc. can be used to specify and model only FRs, whereas, our introduced NFR-process model within the proposed architecture can be used to specify and realize NFRs. According to proposed architecture, FR and NFRs are implemented separately in business logic layer and aspect layer, respectively. This technique is envisaged to assist maintaining the system easily and straightforwardly.

The paper is organized into six sections. This section, being the first section, offers a brief overview of the need for separating NFRs from FRs for robust software development. Section II provides an overview of existing NFRs techniques, methods and frameworks and Section III critically evaluates existing techniques. Proposed architecture for separation of NFRs and NFR-process model is provided in section IV. The next section presents a case study for validation of the proposed methodology. Section VI summarizes conclusion and possible future dimensions of the research.

**LITERATURE REVIEW**

This study takes into account general NFRs techniques as well as particular approaches and methods for the NFRs attributes specified in FURPS (functionality, usability, reliability, performance and scalability) quality attribute model (Chung & Supakkul, 2006). Techniques to model and analyze NFRs from the concept of software product line using Extended PLUS (Product Line UML-Based Software Engineering) are use by (Nguyen, 2009) which specifies performance requirements (Chung et al., 2000). However, the Extend PLUS approach is only applicable to performance requirements. (Ernst & Matalkah, 2006) highlight a generalized concept of visualization by associating visual variables (color, shape, size) with various quality attributes in goal-oriented model for requirement engineering. However, the proposed
framework is not feasible for large systems and fails to address empirical validation for the framework. (Mairiza, 2010) highlights quantitative concepts of NFRs with respect to definition and terminology, types and NFRs in different types of systems and application domains. Whereas, structural hierarchy of NFRs type, as used in other quality attribute models such as McCall, Boehm or ISO 9126 (Rawashdeh & Matalkah, 2006), is also required in such frameworks. (Kassab et al., 2007) propose extension of the existing NFR framework by adding few notations to specify NFRs in SIG (Softgoal Interdependency Graph) and prioritize the extended NFRs framework with MAUA (Multi-Attribute Utility Assessment) technique as well as identify potential risks in software development by elucidating COSMIC-FFP (Common Software Measurement International Consortium - Full Function Point) technique. Further research is required to determine the impact of employing such techniques on the development cost.

A detailed survey of the existing definitions of the NFR terms and problem associated with them has been conducted by (Glinz 2007). (Herrmann et al., 2007) discuss and compare MOQARE (Misuse-Oriented Quality Requirements Engineering) and IESE-NFR (Fraunhofer Institute for Experimental Software Engineering NFR) methods to derive requirements from quality attributes. More research probes are required to devise methods for obtaining occurrence probabilities of threats along with their severity as in by (Alexander 2003). (Zhu & Gortan, 2007) propose a UML profile for modeling design decisions and NFRs. (Kassab et al., 2009) propose a meta-model to design and trace NFRs. However, due to the complexity of the relations in traceability model, a change analysis mechanism is required to ensure consistency of a particular change.

(Glinz 2007; Rosa et al., 2000) discusses the problems faced in notions of NFRs and proposes new systematic definitions for the terms that are normally ambiguously used. (Fei & Xiaodong, 2007) propose the XML-NFR technique to model NFRs using XML by discussing different techniques for individual NFRs specified in FURPS model, but it is imperative to import quantitative index of NF-attributes into XML-NFR model for validation of the model.

Functionality attribute to capture and reuse functional and non-functional requirements has been studied by (Chung & Supakkul, 2006; Cyneiros 2001; Gross, D & Tu, 2001) by formulating a goal-object pattern framework which requires tool support to structure patterns in a repository. Usability aspect is studied at length by (Heiskari et al., 2009) who identify gaps between usability and RE by highlighting usability in different roles; and (Chung & Sampaio-do-Pradoleite, 2009) stress that user studies play an active role in usability attribute
of NFR. However, quantifying the usability requirements is always a challenging task. Various techniques for collecting data for software reliability estimation are discussed by (Dimov et al., 2010), but such techniques require meticulous empirical validation for the feasibility of mutation testing on software reliability estimates. Likewise, (Prasad et al., 2009) propose a sequential Bayesian approach for measuring software reliability by using Operational Profile for quantitative characterization of how the software is expected to be used; but, Operational Profiles are computationally intensive and expensive to apply. (Wei Ho, 2007) propose software PREM (Performance Requirements Evolution Model) for performance requirement specifications and validation. Such models entail that criteria to stop the requirement refinement process should be clearly outlined (Dobson et al., 2007). Software maintainability/scalability classification is studied by (Rawashdeh & Matalkah, 2006; Huang 2005), but it is necessary to supplement such techniques by developing a checklist on classification of maintainability requirements.

CRITICAL EVALUATION

(Nguyen 2009) proposed extended plus approach in requirement modeling. In the support of (Nguyen 2009; Ernst et al., 2006) provided a visualization of NFRs, which associates different visual variables (color, shape, size or hue) with various quality attributes and depicted them in goal-oriented model. (Kassab et al., 2007) proposed an extension to NFR framework. The technique explained by (Ernst et al., 2006) helps to determine the problems when requirements are changed by means of OpenOME - (a requirement engineering tool) whereas; (Fei & Xiaodong, 2007) proposed XML-NFR technique to model NFRs using XML. The study conducted by (Mairiza et al., 2010) elaborates the importance of NFRs to visualize NFR’s terminologies through content analysis; while, (Glinz 2007) discussed the problems faced in notions of NFRs and proposed the solutions to overcome highlighted problems. (Kassab et al., 2007) introduced a function ‘M’ (for mapping) to identify the risks; whereas, the technique MOQARE discussed by (Herrmann et al., 2007) analyzed NFRs as business goals, quality deficiencies and misuse case which are later depicted in the form of a misuse tree. (Zhu 2007) proposed a UML profile for modeling design decisions and NFRs. In support of (Zhu 2007), (Kassab et al., 2009) proposed a meta-model to design and trace NFRs. In addition, (Chung & Supakkul, 2006) proposed a goal-object pattern framework to capture and reuse functional and non-functional requirements. (Dimov et al., 2010) collects the data for software reliability estimation through mutation testing whereas; (Prasad et
al., 2009) suggested that reliability of software depended on how the system would be used. Performance is given great importance in, (Nguyen 2009; Ernst et al., 2006; Kassab et al., 2007; Zhu 2007; Glinz 2007; Fei & Xiaodong, 2007; Chung & Supakkul 2006 and Chung.2000). In the support, (Wei Ho 2007) proposed software performance requirements evolution model (PREM) for performance requirement specifications and validation. Further, (Heiskari et al., 2009) discussed usability in different roles such as user studies and requirement reviews while (Nguyen. 2009) describes usability requirements in use case description.

**Architecture for Separation of NFRs and FRs**

A limitation observed during the literature review of this research is that these techniques proposed models which usually integrate FRs and NFRs, but when these models are implemented, they creates interdependency between these two requirements (Huang, 2005). For example, in an application, if there is any change in functionality of the system then it may affect system's scalability, performance or some other NFR. Since, no such methodology is proposed earlier for separation of FRs and NFRs, therefore, we have endeavored to address this issue in our study.

To handle aforementioned problem, we propose Model View Controller based architecture to separate NFRs from FRs during implementation as depicted in Figure 1. The architecture is composed of four layers - presentation layer, business logic layer, aspect layer and data access layer - as explained below. Splitting an application into tiers will assist developers to modify a specific layer rather than to rewrite the entire application. In addition, it enables the distribution of application functionality across different independent systems. These tiers are logical tiers and may not necessarily be running on same physical server. For example, end user components may be running on local workstations (first-tier), applications processes may be running on remote servers (second-tier) and a collection of databases maybe store in another server (third-tier). The detailed description of the different components of our proposed framework along with their functionality is provided below.
**First tier- presentation layer:** This is the topmost level of the application. The presentation layer displays e-commerce, library data to the user as well as, it communicates with the business logic layer for requested information. End users do not access the third-tier services directly. For example, first-tier provides a form on which users order a product in e-commerce application. First-tier submits this order to the second-tier processes, which check the product databases and execute the tasks that are needed for billing and shipping.

**Second Tier- Business Logic layer and Aspect Layer:** The second-tier processes are usually referred to as the business logic layer (BLL). These processes manage the application and can directly access to third-tier services. Most of the processing occurs in the BLL. For example, if customers place an order in e-commerce application, BLL will update purchase database or inform the customers with un-availability of an item. Separating the second and third tiers reduces the overhead on the third-tier services, supports more effective connection management and may improve overall network performance.

Our architecture proposes aspect layer (AL) along with BLL in second tier. BLL is used to implement FRs for example, place order, register course or deposit cash
whereas, AL is responsible to implement NFRs such as performance, security or traceability aspects. NFRs can further be classified as well, for example, performance can be categorized as response time and throughput or traceability can be categorized as logging and so forth. However, FRs in BLL and NFRs in AL will both communicate with each other during the implementation. The communication between these two layers is handled by XML file during implementation which determines that a FR such as "borrow book" on BLL needs to have "traceability" requirements in AL. To demonstrate this communication we have implemented ZABLibrary system in Spring AOP as illustrated in the subsequent section.

**Process model for FRs:** There are various software process models for software development. Mostly process models go through following common phases such as requirement, design, implementation etc. The goal of this architecture is separation of FRs and NFRs during software development. Therefore, we consider applying existing process models for specifying and modeling FRs only, as existing process models can provides a standardize development of FRs. Later these requirements can be implemented in business logic layer.

**Process model for NFRs:** We need to consider different aspects within a system, for example, performance, security etc. To handle these aspects/NFRs we propose a process model to specify and model them separately. NFR-process model undergoes through the following processes, as explained below:

1. **Identify and Classify NFRs:** Requirements emerge through a dynamic social process and are usually expressed in natural language. Once requirements are specified by user, analysts identify and classify NFRs concerning different aspect of software. Identification of NFRs is a process of delineating constraints in the system such as performance, security etc; whereas, classification deals with categorization of NFRs. There are a number of quality models such as Mc Call, Bohem (Rawashdeh et al., 2006) each one of these quality models can assist in classification of NFRs. In Figure 2, we provide a representation of identified NFRs and their classification that will further proceed to realization. In addition, these requirements are organized into a template later on.

![Image of a figure showing the identification and classification of NFRs.](image-url)
2 - **Realization of NFRs in UML**: UML element allows the requirements to be traced as well as it can be used to model or document any requirements ranging from FRs through to performance or security requirements. Realization of NFRs in UML provides a relationship between one NFR to another NFR’s behaviors. Further, these NFRs can be quantified so that they can be easily understood by the developer.

3 - **Implementation of NFRs in Aspect Layer**: NFRs bear more risks than functional requirements, as they are difficult to implement. But the emerging aspect-oriented techniques offer a promising approach for capturing and implementing the NFRs. The main advantages to apply aspect-oriented techniques to NFRs are to decrease the complexity of software by concentrating on different concerns separately such as performance, security, error logging etc. as well as to improve the modularity of software artifacts.

**Third Tier- Data Access Layer (DAL)**: The third-tier services are protected from direct access by first tier. This layer consists of data access components to assist resource sharing and allow end users to configure without installing the DBMS libraries or drivers. In addition, it supports and manages database and is responsible for storing and retrieving application data. This layer only entertains requests from the BLL and validates the information provided by the user.

**Data Source**: It could be a database or data warehouse that is responsible for storing data. Database may contain text-files, XMLI-files etc. Within the architecture, the database for FRs is also separated from database of NFRs. This will help to reduce the complexity of software and maintain the system.

The main advantage of using this architecture is separation of requirements as each layer has a pre-defined functionality. Introducing aspect layer increases flexibility of the architectural model. The architecture is able to handle frequently changing business requirements which are likely to be changed with introduction to new policies and services being offered. The architecture allows the systems to be loosely coupled so whenever any new FR or NFR is added onto the system, the specific layer for that requirement is changed only without affecting rest of the system layers. As a result, this architecture makes the maintenance of the system easy. In addition, separate databases for FRs and NFRs provide efficient query processing which increase system performance.

**Case Study**

While proposing this architecture, we consider the example of developing a library system ‘ZabLibrary’ for SZABIST (our institute) as a case study. This system will be capable to contain a rich collection of books on information
technology and management sciences topics. In addition, the ZabLibrary will also be able to subscribe to different online libraries through which students could access online e-books, computer journals and magazines.

**Problem Statement:** Borrowing books, returning books or viewing the available books at the library is currently done manually where a student has to go to the shelves and check for the availability of the books. The librarian then checks the student ID and issues him/her a specific book. Librarian has to update the members and books database manually. Other factors that make the system inefficient include:

- **Time:** Manual systems consumes sufficient amount of time that could be otherwise spent in performing some other useful work.

- **Delays:** The problem mentioned above also leads to serious delays in searching a book by title or author name etc.

- **Excessive Load Work:** Since most of the work is done manually, so the users of the current system mostly remain very committed and overloaded with the amount of work. This greatly reduces the efficiency and also the quality of the results produced.

Two major flaws of the existing system are time delays and excess of load work. The success of ZabLibrary will be measured on the basis of how well these two factors are eliminated.

To overcome this problem, we decided to develop an Online Library Management System named ZabLibrary. This system would be used by members of library within the university to check the availability of the books, borrowing the books, help librarian to update the databases and various other features. A small part of FRs and NFRs is considered for ZabLibrary in this paper. Based upon these requirements we will further explain our introduced NFR-process model as: “A web-based Library Management System needs to be developed where users have to enter login-id and password to access the system. ZabLibrary is used by students, instructors and librarian. Students can view, search and borrow a book. Faculty members can view and download materials related to book, e.g. lecture slides, notes etc. whereas, librarian has all the access rights to use the system. In addition, each activity performed by any user needs to be logged and users must be validated for the access rights they have in the system. ZabLibrary must be available 24/7 and should be capable of concurrent processing of multiple user requests.”

**First tier:** The presentation layer displays ZabLibrary information to the students, faculty etc. Users can access the system from the Internet using HTML or its derivative technologies as the system uses a web browser as an interface.
The system is quite user friendly and self-explanatory, hence, users can easily operate it.

**Second Tier- Process model for FRs:** In ZabLibrary we separated FRs and NFRs by using two different process models simultaneously. Conventional SDLC process model was applied to specify and model FRs. However, user requirements were employed to identify FRs and we subsequently modeled it in UML diagrams. Later, we implemented FRs only in business logic layer creating a separate database for these requirements.

**Second Tier- Process model for NFRs:** In order to identify and modeled NFRs, we applied our proposed NFR-process model to ZabLibrary system. These requirements were later implemented in Aspect layer.

**Identify and classify NFRs:** We elicited our NFRs by disturbing a questionnaire to stakeholders. In addition, we classified NFRs and organized them into template so that they can be easily maintained. Performance is a vast term and it needs to be classified to understand the real requirements of stakeholder, therefore, we organized different NFRs into a template as depicted in Table 1. Whereas, Table 2 depicts an example of performance classification in ZabLibrary system.

<table>
<thead>
<tr>
<th>NFR</th>
<th>The name of Non-functional Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Categorization of NFRs</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>Associated stakeholders with a specific NFR</td>
</tr>
<tr>
<td>Description</td>
<td>Intended behavior of the NFR</td>
</tr>
<tr>
<td>Priority</td>
<td>Expresses the importance of the NFR for the stakeholder such as 1, 2, 3; where 1 being the highest and 3 the lowest. Note: NFRs Prioritization play a significant role in reducing requirement problems and increasing customer satisfaction. The most important NFRs are numbered 1 which means these NFRs are essential, non-negotiable and are required right now. The ones which are negotiable or can be delayed a little later are numbered 2 and those that are desirable, flexible are number 3. However, priority attribute is handled in the aspect layer by maintaining requirement information throughout the project. As soon as analysis and design issues are resolved, the priority of NFRs needs to be updated. Likewise, when new requirements are introduced a reassessment of priority shall be done.</td>
</tr>
<tr>
<td>Assumptions</td>
<td>List the factors that affect these NFRs</td>
</tr>
<tr>
<td>Dependencies</td>
<td>List of requirements on which this NFR is dependent.</td>
</tr>
</tbody>
</table>
Table 2. Performance Template.

<table>
<thead>
<tr>
<th>NFR</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classification</strong></td>
<td>Response time, throughput, speed and accuracy of processing</td>
</tr>
<tr>
<td><strong>Stakeholders</strong></td>
<td>Student, Librarian, Faculty Members</td>
</tr>
</tbody>
</table>
| **Description** | Every webpage should be downloaded in 15sec or less over a 50 kbps modem connection.  
The information is automatically refreshed every two minutes.  
The system is capable of handling 250 users at a time  
The computer involved should have fast response time for 2 seconds. The product needs to be hosted on a web server; therefore, delay in the network should be kept less than 3 seconds  
The system shall be available 99.5% of the time  
The system shall take as less than 2 seconds to provide service to the administrator or the librarian.  
The number of transactions is directly dependent on the number of users; the users may be the students, librarian, faculty members and also the people who use library for checking-out books, returning books and checking online library accounts.  
The system shall respond to the member in not less than two seconds from the time of the request submission. The system shall be allowed to take more time when doing large processing jobs. |
| **Priority** | 1 (High) |
| **Assumptions** | The university computers should have Internet connection and Internet server capabilities  
The existing Local Area Network (LAN) will be used for collecting data from the users and also for updating the library catalogue |
| **Dependencies** | Microsoft SQL server should be used to store the library database.  
Performance requirements are stored in NFRs database. |

**Realization of NFRs in UML:** Existing models usually associate UML diagrams with NFRs representation such as goal-framework (Chung & Supakkul, 2006). This results in cluttered diagrams which are difficult to understand. In addition, there are different quality attributes models such as McCall, Boehm etc. (Rawashdeh & Matalkan, 2006) but we created realization of NFRs in UML based upon requirements in ZabLibrary as depicted in Figure.3.
Fig. 3. Realization of NFRs in UML for ZabLibrary System.

Highest prioritized requirements for ZabLibrary which were identified from the template are specified in NFRs followed by other requirements such as performance, logging, etc. This model shows classification of NFRs as well.

Further, we have quantified these NFRs to facilitate developer to understand the system at length. Figure 4 depicts quantified NFRs for ZabLibrary.

Fig. 4. Quantified NFRs for ZabLibrary System.

**Implementation:** In this section, we show the implementation and communication for “borrowing book” in BLL layer and “logging” in AL features from ZabLibrary.
system to provide the proof of concept by using Spring AOP. Borrowing book functionality requires user to enter name, id and book name. Logging aspect records the time and functionality performed by the user. We have imported two main libraries org.apache.log4j.Logger and org.apache.log4j.Priority for logging aspect.

This paper demonstrates the development of 2nd tier only in the proposed architecture which is development of BLL and AL and communication between these two layers. As mentioned earlier, integration of FRs and NFRs during development creates interdependency between these two requirements. Therefore, in Figure 5, we depict how it influences each other when FRs and NFRs are implemented together.

```java
package org.lms.simple;

import java.util.Scanner;
import org.apache.log4j.Logger;
import org.apache.log4j.Priority;
import org.springframework.util.StopWatch;

public class BorrowBook {
    String userName;
    int userId;
    String bookName;

    public void BorrowingInformation() {
        Scanner in = new Scanner(System.in);

        System.out.println("Please Enter UserName : ");
        userName = in.nextLine();

        System.out.println("Please Enter BookName : ");
        bookName = in.nextLine();

        System.out.println("Please Enter UserID : ");
        userId = in.nextInt();

        System.out.println("NOTE: 'Book needs to be RETURNED within 7DAYS' ");
    }

    public void MaintainBookLog() {
        Logger logger = Logger.getLogger(BorrowBook.class);
        logger.log(Priority.INFO, userName + " has accessed the LMS .......");
        StopWatch clock = new StopWatch("Book Log has been maintained for UserName: " + userName + " with ID: " + userId + "; BookName: " + bookName + ";
        System.out.println(clock.shortSummary() + "\n");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        BorrowBook bb = new BorrowBook();
       (bb.BorrowingInformation());
       (bb.MaintainBookLog());
    }
}
```

Fig. 5. Integration of FRs and NFRs during development.

The method BorrowingInformation() seeks user’s inputs to borrow a book. In addition, the method MaintainBookLog records the time and functionality
performed by the user. The method `MaintainBookLog()` records the time by importing a `stopwatch` class - a utility provided by Spring AOP framework; whereas, class `logger` and `priority` in `log4j` container is used for tracking the information performed by the user.

Supposedly, if this `MaintainLog()` needs to be attached throughout the system, we need to copy this code again and attach it with some another functionality. As a result, business logic layer will be all lost and it will really be difficult to identify FR's and NFR's implementation code. Alternatively, if we create separate classes for `borrowing book` and `MaintainBookLog`, then it would create interdependency between the classes. As depicted in the `Main()` method, the object of `BorrowBook` is accessing the method `MaintainBookLog`. To permeate this problem, we implemented NFRs separately from the business logic in aspect-oriented programming. Implementation process consists of three major steps:

a - Creating Business logic layer and aspect layer
b - Handling communication between BLL and AL
c - Generating Driver class

a - Creating Business logic layer and aspect layer: We have separated FRs which is `Borrowing Book` functionality in business logic layer and NFRs which is `Maintain Book Log` in aspect layer as depicted in Figure 6 and Figure 7.

```java
package org.lms.simple;
import java.util.Scanner;

public class BorrowBook {
    String userName;
    int userId;
    String bookName;

    public void BorrowingInformation(String UserName, int id, String BookName) {
        Scanner in = new Scanner(System.in);

        System.out.println("Please Enter UserName : ");
        userName = in.nextLine();

        System.out.println("Please Enter BookName : ");
        bookName = in.nextLine();

        System.out.println("Please Enter UserID : ");
        userId = in.nextInt();

        System.out.println("NOTE: 'Book needs to be RETURNED within 7 DAYS'");
    }
}
```

**Fig. 6.** Implementation of `Borrow Book` in Business Logic Layer.
package org.lms.aop;
import org.apache.log4j.Logger;
import org.apache.log4j.Priority;
import org.springframework.util.StopWatch;

// Aspect Class
public class LoggingAspect {
    public void MaintainBookLog(String UserName, int id, String BookName) {
        Logger logger = Logger.getLogger(LoggingAspect.class);
        logger.log(Priority.DEBUG, "UserName " + UserName + " has accessed the LMS ....... ");
        StopWatch clock = new StopWatch("Book Log has been maintained for UserName: '" + UserName + ", id: " + id + ", BookName: '" + BookName + "'");
        System.out.println(clock.getShortSummary());
    }
}

Fig. 7. Implementation of Logging in Aspect Layer.

b - Handling communication between BLL and AL: To handle communication between these two layers, an application context.xml file is added in the system as depicted in Figure 8.

<bean id="Borrowbook" class="org.lms.aop.Borrowbook"/>
<bean id="loggingAspect" class="org.lms.aop.LoggingAspect"/>

<aspect ref="loggingAspect">
    <pointcut id="BookLogAspect" expression="execution(* org.lms.aop.Borrowbook.BorrowingInformation (String,int,String)) and args(UserName, id, BookName)="/">
        <around pointcut-ref="BookLogAspect" method="MaintainBookLog"/>
    </pointcut>
</aspect>
</beans>

Fig. 8. Snapshot of Context.xml - for handling communication between BLL and AL.

Firstly, we need to specify class path for both layers (BLL and AL) in Bean id. Bean is a representation of classes in XML. <aop:Aspect-ref = "loggingAspect"/> is passing a reference to beannid = "loggingAspect" to get the class path of aspect layer. <aop:Pointcut-id /> is the event when an aspect (logging aspect) needs to be executed. In our case, as class borrowbook with method borrowinginformation is called the logging aspect will be executed. In aspect layer, there can be more than one aspect. However, <aop:around-pointcut-ref /> specify the concerns which needs to be executed within aspect. We execute the logging aspect onto the class borrowbook - the classes which are totally independent of each other and
communication between these two classes is handled outside a class in context.xml file. Hence, if we wish to make any change in functionality of the system, NFRs in aspect layer will not be affected.

c - Generating Driver class

We have imported two classes BeanFactory and ClasspathXMLApplicationContext to load XML file in Spring AOP as depicted in Figure 9. A single object ctx is created to handle the functionality in both classes. This object is responsible to handle beans in XML file and to load context.xml.

```java
package org.lms.aop;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public final class Main {

    public static void main(String[] args) {
        BeanFactory ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
        BorrowBook bb = (BorrowBook) ctx.getBean("Borrowbook");
        bb.BorrowingInformation();

        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

**Fig. 9. Driver Class for LMS.**

Further, ctx is getting bean borrowbook from context.xml file and is casting this bean id to java object. As a result, the class borrowbook’s object is created irrespective of importing its class initially.

**Output:** A screen shot for ZabLibrary is illustrated in Figure 10.
5.4 Data Access Layer (DAL): SZABIST has installed state-of-the-art computer facilities. IBM Netfinity 5100 Server is specifically used for ZabLibrary system to support, manages and validates user requests.

5.5 Data Source: Microsoft SQL server is used to store the database. The database for FRs is also separated from database of NFRs for the system. This assists to maintain the system. The proposed architecture can be implemented in any operating systems or database management systems. The choices made in section 5.4 and 5.5 demonstrate that the system implemented on the proposed architecture can be installed on any available hardware or DBMS. At the same time, it is also cost effective, as the system can be launched with a reduced investment and later could increase the performance with a light improvement of the equipments or software versions.

6.6 Results and Discussions: NFRs are normally neglected in software applications. We tackle this problem by treating NFRs in separate layer so that they can be considered as a vital part of software development process. We also introduce a separate process model within the architecture to organize NFRs into template so they can be easily maintained. Developing NFRs in aspect layer let us reduce the interdependency between the classes during the implementation and if any requirements are changed, it does not affect the entire system. In addition, during implementation we do not have to copy NFR code and attached it with some other functionality which also reduces code redundancy. As a result the implementation code for FRs and NFRs can be effortlessly identified.
In the last twenty years, both the functionality and technology of off-the-shelf software systems have significantly improved. Moving to a new application provides more intuitive, easier to use system, broader access to information and greater flexibility to meet market’s evolving needs. In our scenario, users expect nearly continuous and real-time access to ZabLibrary systems which was not possible by existing system with time delays and excess of load work. In addition, requirements are changing on daily basis in world of software engineering. This becomes ad-hoc for developers to maintain the system if requirements are not clearly stated or implemented. The modularization of software becomes necessary with this rapid change of requirements. In addition, the system is refined by more operational features day to day which are easily integrated. As the system is maintained by few administrative staff members, therefore, it offers a cost effective solution. The limitation faced during the implementation of ZabLibrary system was that there are few automated testing tools for NFRs are available and most of the automated testing tools focus testing FRs only, such as JUNIT.

**Conclusion and Future Work**

The papers proposed architecture to separate NFRs. The proposed architecture adds aspect layer in 2-tier to support NFRs and introduces a process model within the architecture to identify, realized and implement NFRs. This architecture is flexible to handle frequent changing requirements in different layers without affecting entire system and allows for effective system maintaining. The implementation of 2\textsuperscript{nd}-tier (business logic and aspect layer) for a library system is accomplish by using Spring AOP. We evaluated our proposed architecture through a case study of ZabLibrary system. The results of the study were promising as we successfully separated NFRs from FRs without any tangible overhead. The proposed architecture also offers a cost-effective solution and can be implemented in iterative fashion. The architecture is capable to support smooth shifting to a new application as it provides an easy to use method.

Our future work has two main directions. First, we will model NFRs and transition it to a realization. Second, since, there are limited numbers of testing tools available to support NFRs; therefore, as a prospective future dimension to this research, we intend to develop a framework for an automated testing tool specifically for NFRs to test system performance, reliability, etc.

**REFERENCES**


Submitted : 05/09/2011
Revised : 23/10/11
Accepted : 25/10/2011
 إطار عمل لفصل العناصر غير العاملة للمحافظة على البرمجيات

محروك عمر و محمد نسيم أحمد خان
معهد الشهيد ذو الفقار علي بوتو للعلوم والتكنولوجيا
إسلام آباد - باكستان

خلاصة

هندسة الضرورة (المطلوبة) جزء حيوي من تطور دورة البرمجيات وذلك لأن جوهر أي برنامج ناجح هو فهم متطلباته. عند تطوير أي برنامج تعتبر الشروط غير العاملة (NFRS) واحد من الأمور التي يصعب التعامل معها، ومع ذلك فإنه لا زال هناك نقص في معرفة طرق نمذجة وتشكيل (NFRS) لأنها عادة - غامضة التعريف بالنسبة للمستخدمين.

هذه الورقة تحلل تقنيات وأطر عمل وتوصيات من أعمال منشورة سابقاً، كما وتبين هذه الورقة محدودية هذه التقنيات بهدف التحليل النقدي لعدة تقنيات (NFRS). زيادة على ذلك فإن الورقة تعرض أسلوباً لفصل (NFRS) عن (FRS) خلال تطوير البرمجة وذلك لتحسين أداء البرمجيات. وبالإضافة إلى ذلك فإن الورقة تقدم نموذج عمل من خلال الأساليب لتشخيص وفهم وتطبيق (NFRS) على وجه الخصوص. الأساليب المقترحة ونموذج عمل (NFRS) تم تكملته بإتاحة فاعليته من خلال دراسة ميدانية على نظام مكتبي.
الجلة العربية للعلوم الإنسانية

بحث باللغة العربية
بحث باللغة الإنجليزية
مناقشات وندوات
عروض البحث الجديدة
تقارير ومؤتمرات

مجلس
نشر
العلمي

رئيسة التحرير
أ.د. سعاد عبدالوهاب العبدالرحمن

P.O.Box: 26585 - Safat. 13126 kuwait
Tel: (+965) 4817689 - 4815453 Fax: (+965) 4812514
E-mail: ajh@kuniv.edu.kw  http://www.pubcouncil.kuniv.edu.kw/ajh/