**Software Requirement Specification**

**Modules:**

**Admin:** This module has the entire access to all other modules, admin creates the project and assigning the projects to the created manager, adding members to the managers, assigning Defects based on the priority. Can update the manager, members and access to the particular project data. Generating reports based on the managers report submission.

**Manager:** Manager has the full access to the particular project assigned by the admin and controls the team members access to the Defects assigned. Has the permission to generate the reports and update the information of team members and adding members to the project.

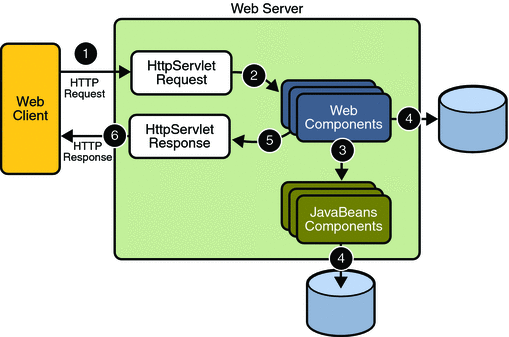
**Developer:** Can access the task or Defect assigned by the manager, view assigned projects and resolving the assigned Defect. Developer can view the Defects list assigned by the manager.

**Tester:** Tester can access to the projects , can view the assigned projects and can add a new Defect to the list and send the bug back to the manager. Tester can login to the system and access the assigned projects list.

**Reports:** Both Admin and Manager can access this module and generate the reports based on the requirements.

**PROCESS FLOW:**

**Architecture Diagram**



1. **THE PRESENTATION LAYER**

Also called as the client layer comprises of components that are dedicated to presenting the data to the user. For example: Windows/Web Forms and buttons, edit boxes, Text boxes, labels, grids, etc.

1. **THE BUSINESS RULES LAYER**

This layer encapsulates the Business rules or the business logic of the encapsulations. To have a separate layer for business logic is of a great advantage. This is because any changes in Business Rules can be easily handled in this layer. As long as the interface between the layers remains the same, any changes to the functionality/processing logic in this layer can be made without impacting the others. A lot of client-server apps failed to implement successfully as changing the business logic was a painful process

1. **THE DATA ACCESS LAYER**

This layer comprises of components that help in accessing the Database. If used in the right way, this layer provides a level of abstraction for the database structures. Simply put changes made to the database, tables, etc do not affect the rest of the application because of the Data Access layer. The different application layers send the data requests to this layer and receive the response from this layer.

1. **THE DATABASE LAYER**

This layer comprises of the Database Components such as DB Files, Tables, Views, etc. The Actual database could be created using SQL Server, Oracle, Flat files, etc.   
In an n-tier application, the entire application can be implemented in such a way that it is independent of the actual Database. For instance, you could change the Database Location with minimal changes to Data Access Layer. The rest of the Application should remain unaffected.

**SDLC METHODOLOGIES**

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

**The following diagram shows how a spiral model acts like:**



**Fig 1.0-Spiral Model**

**ADVANTAGES**

* Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier.
* It is more able to cope with the changes that are software development generally entails.
* Software engineers can get their hands in and start woring on the core of a project earlier.

**Software Requirements**

Operating System : Windows XP/2003 or Linux/Solaris

User Interface : HTML, CSS

Client-side Scripting : JavaScript

Programming Language : Java

Web Applications : JDBC, JNDI, Servlets, JSP

IDE/Workbench : MyEclipse8.6

Database : Oracle10G

Server Deployment : Tomcat6.0

**Hardware Requirements**

Processor : Pentium IV

Hard Disk : 40GB

RAM : 256MB

FUNCTIONAL

REQUIREMENTS:

**Authentication:**

* Registration
* Login
* Logut
* Changepassword
* Viewprofile

**Maintenance:**

* User Maintenance: Creating, Granting & Revoking access and deleting users from application.
* Component Maintenance: Creating a component (application being developed / enhanced), Granting & Revoking access on components to Users and Marking a component as “Active” or “Closed”.
* Defect Tracking: Creating, Assigning Defects to users, Modifying and Closing a Defect. A Defect screen should at least have following details
* Defect Number and Title
* Defect priority
* Date created

NONFUNCTIONAL

REQUIREMENTS:

Non-functional requirements tend to be stated in terms of **constraints** on the results of tasks which are given as functional requirements (e.g., constraints on the speed or efficiency of a given task), a task-based functional requirements statement is a useful skeleton upon which to construct a complete requirements statement. That is the approach taken in this work. It can be helpful to think of non-functional requirements as adverbially related to tasks or functional requirements: how fast, how efficiently, how safely, etc., is a particular task carried out by a particular system?

Non-functional requirements are often called **qualities** of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements". Qualities, that is non-functional requirements, can be divided into two main categories:

1. Execution qualities, such as security and usability, which are observable at run time.
2. Evolution qualities, such as [testability](http://en.wikipedia.org/wiki/Software_testability), maintainability, extensibility and scalability, which are embodied in the static structure of the software system

The non-functional requirements are

1. Accessbility

2. Availabilty

3. Scalabilty

4. Portability

5. Robustness

**Accessibility** is a general term used to describe the degree to which a product, device, service, or environment is available to as many people as possible. Accessibility can be viewed as the "ability to access" and benefit from some system or entity. Accessibility is often used to focus on people with disabilities or special needs and their right of access to entities, often through use of [assistive technology](http://en.wikipedia.org/wiki/Assistive_technology).

Accessibility is not to be confused with [usability](http://en.wikipedia.org/wiki/Usability) which is used to describe the extent to which a product (e.g., device, service, environment) can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Availabilty

* The degree to which a [system](http://en.wikipedia.org/wiki/System), [subsystem](http://en.wikipedia.org/wiki/Subsystem), or equipment is in a specified operable and committable state at the start of a mission, when the mission is called for at an unknown, *i.e.,* a random, time. Simply put, availability is the proportion of time a system is in a functioning condition. This is often described as a **mission capable rate**. Mathematically, this is expressed as 1 minus [unavailability](http://en.wikipedia.org/wiki/Unavailability).
* The ratio of (a) the total time a [functional unit](http://en.wikipedia.org/wiki/Functional_unit) is capable of being used during a given interval to (b) the length of the interval.

For example, a unit that is capable of being used 100 hours per week (168 hours) would have an availability of 100/168. However, typical availability values are specified in [decimal](http://en.wikipedia.org/wiki/Decimal) (such as 0.9998). In [high availability](http://en.wikipedia.org/wiki/High_availability) applications, a metric known as [nines](http://en.wikipedia.org/wiki/Nines_%28engineering%29), corresponding to the number of nines following the decimal point, is used. In this system, "five nines" equals 0.99999 (or 99.999%) availability.

**scalability** is the ability of a system, network, or process, to handle growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.[[1]](http://en.wikipedia.org/wiki/Scalability#cite_note-0) For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in a [commercial](http://en.wikipedia.org/wiki/Commerce) context, where scalability of a company implies that the underlying [business model](http://en.wikipedia.org/wiki/Business_model) offers the potential for [economic growth](http://en.wikipedia.org/wiki/Economic_growth) within the company.

Scalability, as a property of systems, is generally difficult to define[[2]](http://en.wikipedia.org/wiki/Scalability#cite_note-1) and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking. A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a **scalable system**. An [algorithm](http://en.wikipedia.org/wiki/Algorithm), design, [networking protocol](http://en.wikipedia.org/wiki/Protocol_%28computing%29), [program](http://en.wikipedia.org/wiki/Computer_program), or other system is said to **scale**, if it is suitably [efficient](http://en.wikipedia.org/wiki/Algorithmic_efficiency) and practical when applied to large situations (e.g. a large input data set or a large number of participating nodes in the case of a distributed system). If the design fails when the quantity increases, it **does not scale**.

The concept of scalability is desirable in technology as well as [business](http://en.wikipedia.org/wiki/Business) settings. The base concept is consistent - the ability for a business or technology to accept increased volume without impacting the [contribution margin](http://en.wikipedia.org/wiki/Contribution_margin) (= [revenue](http://en.wikipedia.org/wiki/Revenue) - [variable costs](http://en.wikipedia.org/wiki/Variable_costs)). For example, a given piece of equipment may have capacity from 1-1000 users, and beyond 1000 users, additional equipment is needed or performance will decline (variable costs will increase and reduce contribution margin).

**Portability** in [high-level computer programming](http://en.wikipedia.org/wiki/High-level_programming_language) is the usability of the same [software](http://en.wikipedia.org/wiki/Software) in different environments. The prerequirement for portability is the generalized [abstraction](http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29) between the application logic and [system interfaces](http://en.wikipedia.org/wiki/Interface_%28computer_science%29). When software with the same functionality is produced for several [computing platforms](http://en.wikipedia.org/wiki/Computing_platform), portability is the key issue for development cost reduction.

This article is about portability in itself. The work required to make software portable is described in the article on [porting](http://en.wikipedia.org/wiki/Porting).

In [computer science](http://en.wikipedia.org/wiki/Computer_science), **robustness** is the ability of a computer system to cope with errors during execution or the ability of an algorithm to continue to operate despite abnormalities in input, calculations, etc. Formal techniques, such as [fuzz testing](http://en.wikipedia.org/wiki/Fuzz_testing), are essential to showing robustness since this type of testing involves invalid or unexpected inputs. Various commercial products perform robustness testing of software systems. Robustness is a consideration in [failure assessment](http://en.wikipedia.org/wiki/Failure_assessment) analysis.