1. INTRODUCTION :

1.1 Objective of proposed system :

Defect Tracking for Improving Product Quality and Productivity for Improving Software Reliability is an automated system that can be useful to employees and the managers in any functional organization. Defect Tracking for Improving Product Quality and Productivity gives the facility to define the tasks in the organization and also allows the managers to track the Defects spent by the developer for that particular task. A report generation facility is supported in DTS that allows the managers to analyze which are those skills by employee are utilized and those which are not utilized. project or application. This tool helps employees to document their Defects and analyze.

1.2 Proposed system :

The Proposed system is a browser which is completely related to online system, which provides the centralized database. It stores Defects data and description of the particular Defect data. It can also create reports and documents based on the information in its database.

1.3 Details of project :

**Existing System :**

The existing system consists of entering the details in the Microsoft Excel Sheets for the storing of the data. When a manager needs information of the employee he searches for the specified file in the file system. He opens the file and takes the information. Report Generation done manually by copying the content of the different files into another file. The Manually generated report was then printed.

**Limitations in Existing System :**

* Information retrieval is a very big process.
* Lack of organization of the files may porn to information loss due to accidental deletion of files.
* No security because the files are visible to the users.
* Report generation will be a big task.

**: Advantages over Existing System :**

* The performance is increased due to well designed database.
* Security is increased
* Time saving in report generation
* Easy to update the details

Need of the system :

[Tracking defects](http://searchsoftwarequality.techtarget.com/answer/When-tracking-defects-make-efficiency-the-end-goal) found in testing as a performance metric tends to be demotivating, placing software testers and developers in an adversarial position when they should be in a cooperative position. Instead, if the purpose of [tracking defects is to look for patterns](http://searchsoftwarequality.techtarget.com/tip/Tracking-bugs-effectively-in-continuous-development) in software development, including requirements definition, design and coding, where problems are found on a regular basis, this may provide information around aspects that need improvement in the organization.

1. Technology & Methodology used :
   1. H/W and S/W Requirement:

**Software Requirements**

1. Operating System : Window XP/2003orLinux/Solaris
2. User Interface : HTML, CSS
3. Client-side Scripting : JavaScript
4. Programming Language : Java
5. Web Applications : JDBC, JNDI, Servlets, JSP
6. IDE/Workbench : MyEclipse8.6
7. Database : Oracle10G
8. Server Deployment : Tomcat6.0

**Hardware Requirements**

1. Processor : Pentium IV
2. Hard Disk : 40GB
3. RAM : 256MB

2.2 Back End and Front End :

Back End : Oracle 10g

Front End : java ver 8

2.3 **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.

**3. Design & Implementation :**

**3.1 SYSTEM DESIGN :**

System design of defect tracking for improving product quality and productivity (D.T.S) contains the various types of tables of records that we can understand from the ER-daigarm and Data flow diagram(D.F.D) with the collection of defferent levels DFD 1,2 .

**3.2** **Flow chart :**

##### DATA FLOW DIAGRAMS:

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

**1. Dataflow:** Data move in a specific direction from an origin to a destination.

**2. Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.

**3. Source:** External sources or destination of data, which may be People, programs, organizations or other entities.

**4. Data Store:** Here data are stored or referenced by a process in the System.

**3.3 Data Flow Diagram :** 

**Level 1 :**



**Level 2:**

**Administrator:**









**LEVEL-1:**

**Employee:**





**Authentication:**



3.4  **UML Diagrams**

**Unified Modeling Language**:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

* + User Model View
    1. This view represents the system from the users perspective.
    2. The analysis representation describes a usage scenario from the end-users perspective.
  + Structural model view
    1. In this model the data and functionality are arrived from inside the system.
    2. This model view models the static structures.
* Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

* Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

* Environmental Model View

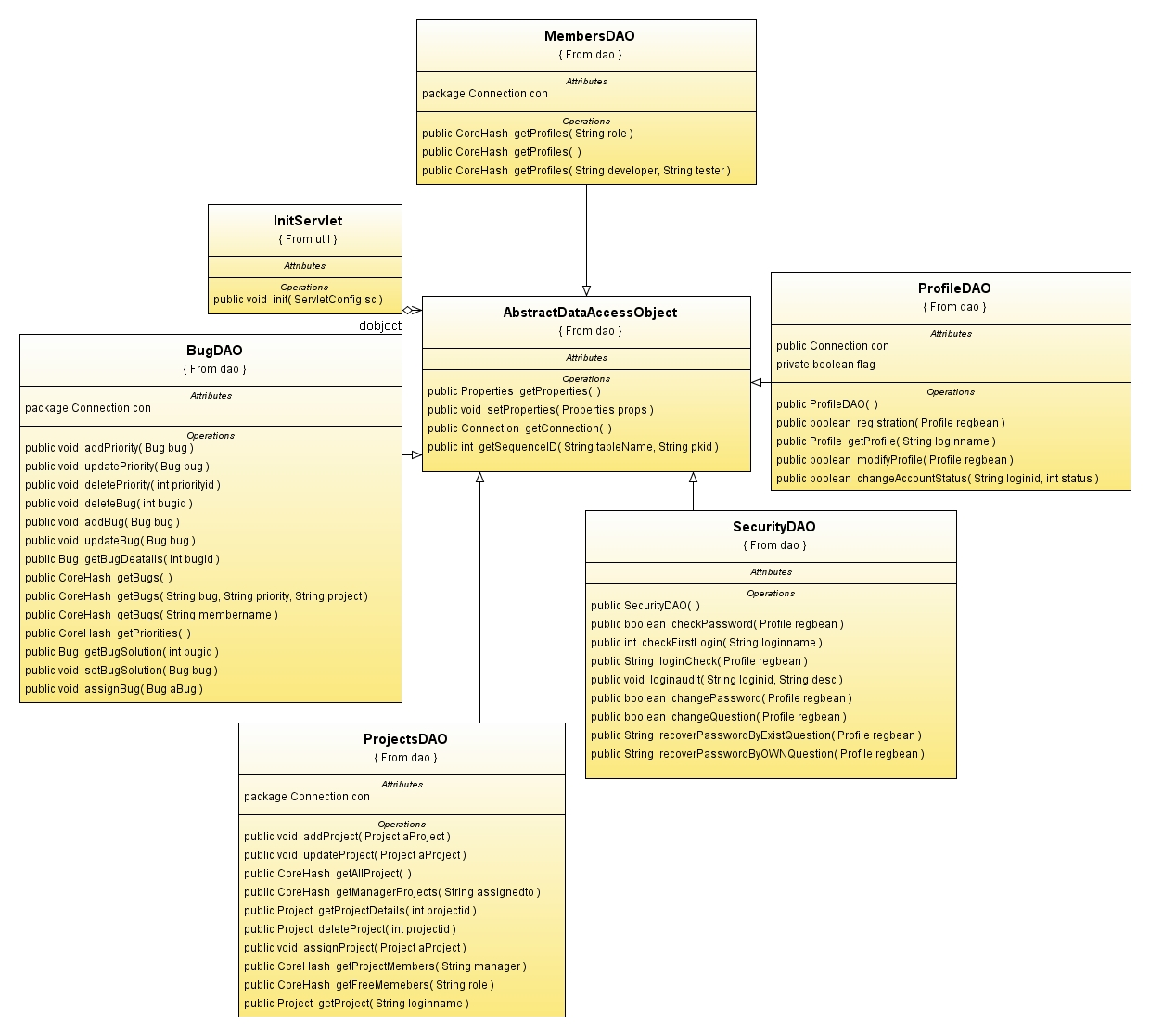
In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

* UML Analysis modeling, this focuses on the user model and structural model views of the system.
* UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

**Class Diagram **

**Admin:** System Usecase Diagram:

System

**Admin**

**Manager**

**Developer**

**Tester**

**Defect Tracking for Improving Product Quality and Productivity**



Manager: 

Developer:



Tester:



SEQUENCE DIAGRAM:

ADMIN:



COLLABORATION DIAGRAM:

ADMIN:



**Manager:**





Developer:





TESTER:



s

Activity Diagrams

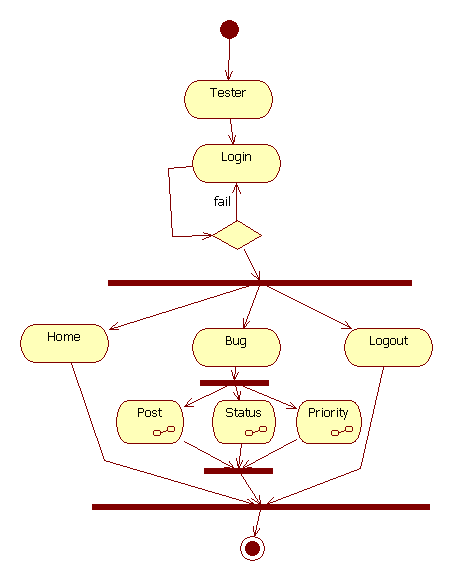
Admin:



Manager:



Tester:



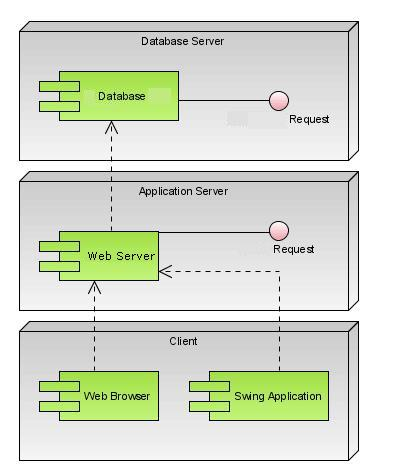
Developer:





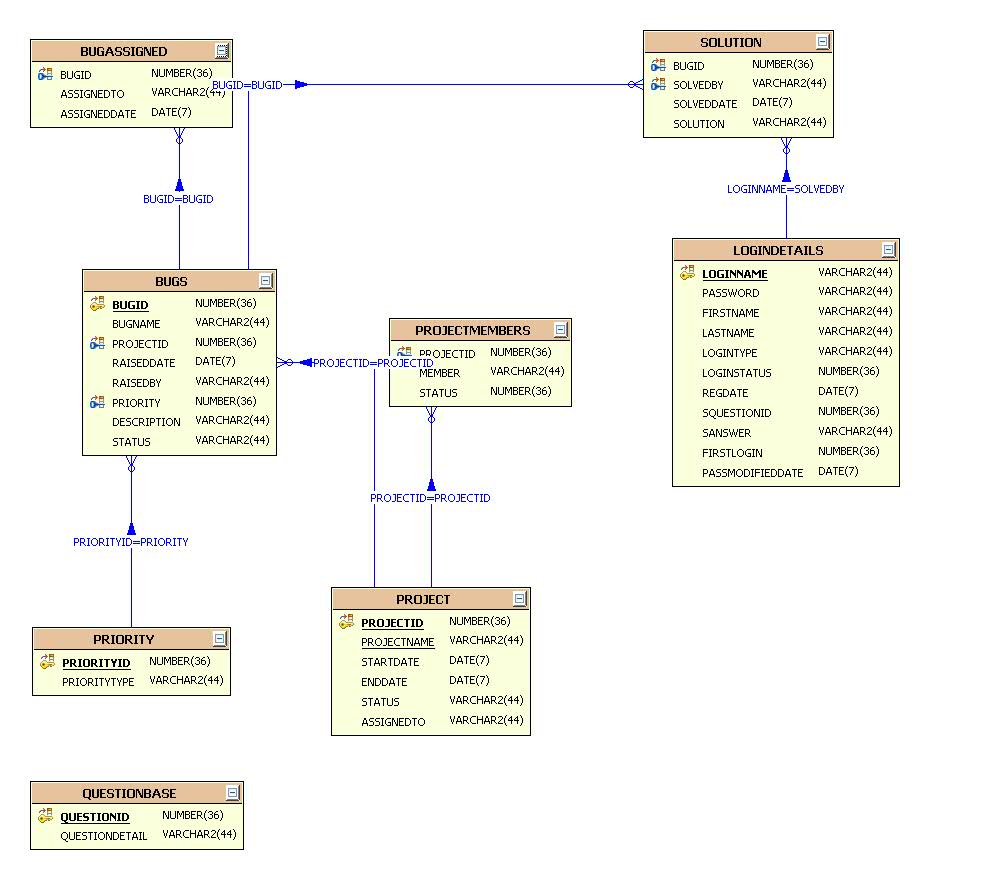
**Component Diagram**

**Deployment Diagram**

****

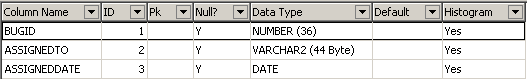
****

ER diagram :

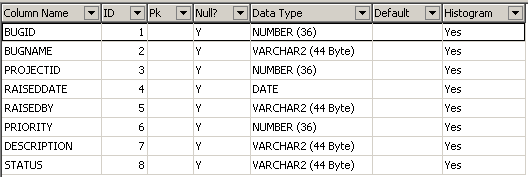


3.6 Tabels :

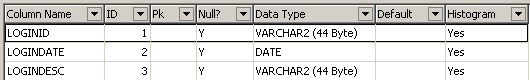
BUGASSIGNED:



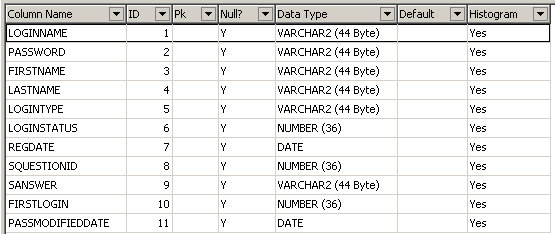
BUGS:



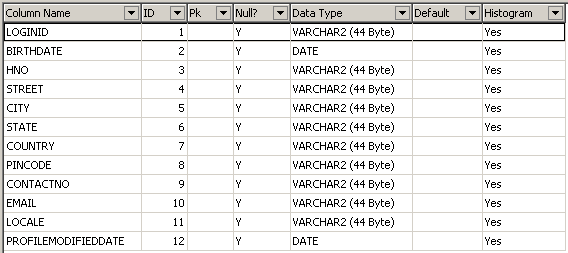
LOGINAUDIT:



LOGINDETAILS:



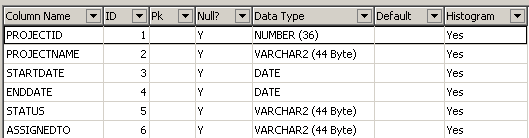
LOGINPROFILE:



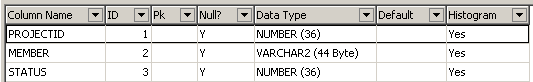
PRIORITY:



PROJECT:



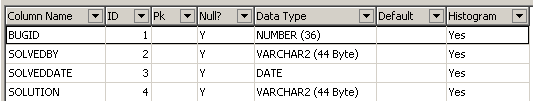
PROJECTMEMBERS:



QUESTIONBASE:



SOLUTION:



**TESTING :**

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification. An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following routine procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behavior in reaction to the probing of the tester[citation needed]. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces. Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes described as one which reveals an error; however, more recent thinking suggests that a good test is one which reveals information of interest to someone who matters within the project community.

**Introduction:**

In general, software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended. Software testing is the technical investigation of the product under test to provide stakeholders with quality related information.

Software testing may be viewed as a sub-field of Software Quality Assurance but typically exists independently (and there may be no SQA areas in some companies). In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in the code or deliver faster.

Regardless of the methods used or level of formality involved the desired result of testing is a level of confidence in the software so that the organization is confident that the software has an acceptable defect rate. What constitutes an acceptable defect rate depends on the nature of the software. An arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than software used to control an actual airliner.

A problem with software testing is that the number of defects in a software product can be very large, and the number of configurations of the product larger still. Bugs that occur infrequently are difficult to find in testing. A rule of thumb is that a system that is expected to function without faults for a certain length of time must have already been tested for at least that length of time. This has severe consequences for projects to write long-lived reliable software.

A common practice of software testing is that it is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes.

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

History

The separation of debugging from testing was initially introduced by Glen ford J. Myers in his 1978 book the "Art of Software Testing". Although his attention was on breakage testing it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Drs. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing as follows: until 1956 it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging. From 1957-1978 there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown, that software satisfies the requirements. The time between 1979-1982 is announced as the destruction oriented period, where the goal was to find errors. 1983-1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a product evaluation is provided and measuring quality. From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults. Dr. Gelperin chaired the IEEE 829-1988 (Test Documentation Standard) with Dr. Hetzel writing the book "The Complete Guide of Software Testing". Both works were pivotal in to today's testing culture and remain a consistent source of reference. Dr. Gelperin and Jerry E. Durant also went on to develop High Impact Inspection Technology that builds upon traditional Inspections but utilizes a test driven additive.

Testing Concepts

* ***Testing***
* ***Testing Methodologies***
* Black box Testing:
* White box Testing.
* Gray Box Testing.
* ***Levels of Testing***
  + Unit Testing.
  + Module Testing.
  + Integration Testing.
  + System Testing.
  + User Acceptance Testing.
* ***Types Of Testing***
  + Smoke Testing.
  + Sanitary Testing.
  + Regression Testing.
  + Re-Testing.
  + Static Testing.
  + Dynamic Testing.
  + Alpha-Testing.
  + Beta-Testing.
  + Monkey Testing.
  + Compatibility Testing.
  + Installation Testing.
  + Adhoc Testing.
  + Ext….

***TCD (Test Case Documentation)***

* ***STLC***
  + Test Planning.
  + Test Development.
  + Test Execution.
  + Result Analysis.
  + Bug-Tracing.
  + Reporting.
* ***Microsoft Windows – Standards***
* ***Manual Testing***
* ***Automation Testing (Tools)***
  + Win Runner.
  + Test Director.

**Testing:**

* The process of executing a system with the intent of finding an error.
* Testing is defined as the process in which defects are identified, isolated, subjected for rectification and ensured that product is defect free in order to produce the quality product and hence customer satisfaction.
* Quality is defined as justification of the requirements
* Defect is nothing but deviation from the requirements
* Defect is nothing but bug.
* Testing --- The presence of bugs
* Testing can demonstrate the presence of bugs, but not their absence
* Debugging and Testing are not the same thing!
* Testing is a systematic attempt to break a program or the AUT
* Debugging is the art or method of uncovering why the script /program did not execute properly.

**Testing Methodologies:**

* **Black box Testing**: is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application.

Usually Test Engineers are involved in the black box testing.

* **White box Testing**: is the testing process in which tester can perform testing on an application with having internal structural knowledge.

Usually The Developers are involved in white box testing.

* **Gray Box Testing**: is the process in which the combination of black box and white box tonics’ are used.

**Levels of Testing:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***Module1*** ***Module2*** ***Module3***   |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |  | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |  | ***Units***   |  |  |  | | --- | --- | --- | |  |  |  | |   i/p ***Integration***  o/p i/p ***Integration o/p*** |   ***System Testing: Presentation + business +Databases***  ***🚹UAT: user acceptance testing*** |

##### STLC (SOFTWARE TESTING LIFE CYCLE)

**Test Planning:**

**1.**Test Plan is defined as a strategic document which describes the procedure how to perform various testing on the total application in the most efficient way.

**2.**This document involves the scope of testing,

**3.** Objective of testing,

**4.** Areas that need to be tested,

**5.** Areas that should not be tested,

**6.** Scheduling Resource Planning,

**7.** Areas to be automated, various testing tools

Used….

**Test Development**:

**1.** Test case Development (check list)

**2.** Test Procedure preparation. (Description of the Test cases).

**1.** Implementation of test cases. Observing the result.

**Result Analysis**: **1.** Expected value: is nothing but expected behavior

Of application.

**2.** Actual value: is nothing but actual behavior of

application

**Bug Tracing:**  Collect all the failed cases, prepare documents.

**Reporting:** Prepare document (status of the application)

**Types Of Testing:**

**🚺>** **Smoke Testing**: is the process of initial testing in which tester looks for the availability of all the functionality of the application in order to perform detailed testing on them. (Main check is for available forms)

**🚺>** **Sanity Testing:** is a type of testing that is conducted on an application initially to check for the proper behavior of an application that is to check all the functionality are available before the detailed testing is conducted by on them.

**🚺>** **Regression Testing:** is one of the best and important testing. Regression testing is the process in which the functionality, which is already tested before, is once again tested whenever some new change is added in order to check whether the existing functionality remains same.

**🚺>Re-Testing:** is the process in which testing is performed on some functionality which is already tested before to make sure that the defects are reproducible and to rule out the environments issues if at all any defects are there.

**🚺Static Testing:** is the testing, which is performed on an application when it is not been executed.ex: GUI, Document Testing

**🚺Dynamic Testing:** is the testing which is performed on an application when it is being executed.ex: Functional testing.

**🚺Alpha Testing:** it is a type of user acceptance testing, which is conducted on an application when it is just before released to the customer.

**🚺 Beta-Testing:** it is a type of UAT that is conducted on an application when it is released to the customer, when deployed in to the real time environment and being accessed by the real time users.

**🚺 Monkey Testing:** is the process in which abnormal operations, beyond capacity operations are done on the application to check the stability of it in spite of the users abnormal behavior.

**🚺Compatibility testing:** it is the testing process in which usually the products are tested on the environments with different combinations of databases (application servers, browsers…etc) In order to check how far the product is compatible with all these environments platform combination.

**🚺Installation Testing:** it is the process of testing in which the tester try to install or try to deploy the module into the corresponding environment by following the guidelines produced in the deployment document and check whether the installation is successful or not.

**🚺Adhoc Testing:** Adhoc Testing is the process of testing in which unlike the formal testing where in test case document is used, with out that test case document testing can be done of an application, to cover that testing of the future which are not covered in that test case document. Also it is intended to perform GUI testing which may involve the cosmotic issues.

**TCD (Test Case Document:**

**Test Case Document Contains**

* **Test Scope (or) Test objective**
* **Test Scenario**
* **Test Procedure**
* **Test case**

This is the sample test case document for the Acadamic details of student project:

**Test scope:**

* Test coverage is provided for the screen “ Acadamic status entry” form of a student module of university management system application
* Areas of the application to be tested

**Test Scenario:**

* When the office personals use this screen for the marks entry, calculate the status details, saving the information on student’s basis and quit the form.

**Test Procedure:**

* The procedure for testing this screen is planned in such a way that the data entry, status calculation functionality, saving and quitting operations are tested in terms of Gui testing, Positive testing, Negative testing using the corresponding Gui test cases, Positive test cases, Negative test cases respectively

**Test Cases:**

* Template for Test Case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Exp | Act | Result |
|  |  |  |  |  |

**Guidelines for Test Cases**:

1. **GUI Test Cases:**

* Total no of features that need to be check
* Look & Feel
* Look for Default values if at all any (date & Time, if at all any require)
* Look for spell check

***Example for Gui Test cases***:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Check for all the features in the screen | The screen must contain all the features |  |  |
| 2 | Check for the alignment of the objects as per the validations | The alignment should be in proper way |  |  |

1. **Positive Test Cases:**

* The positive flow of the functionality must be considered
* Valid inputs must be used for testing
* Must have the positive perception to verify whether the requirements are justified.

***Example for Positive Test cases:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Check for the date Time Auto Display | The date and time of the system must be displayed |  |  |
| 2 | Enter the valid Roll no into the student roll no field | It should accept |  |  |

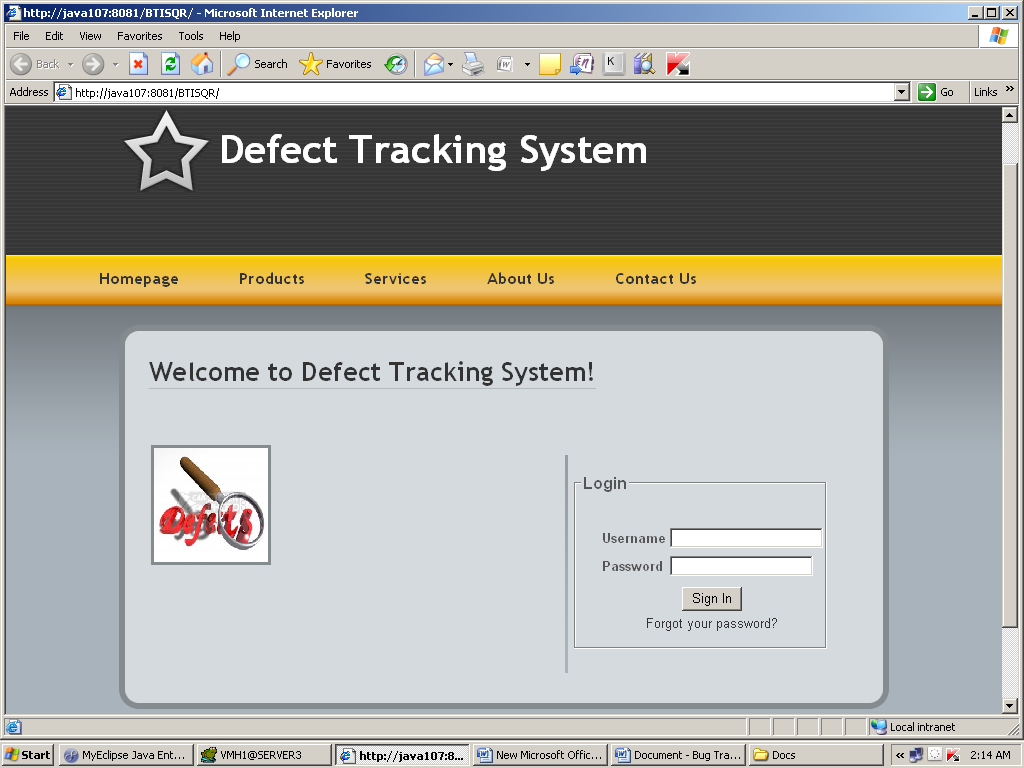
1. **Negative Test Cases:**

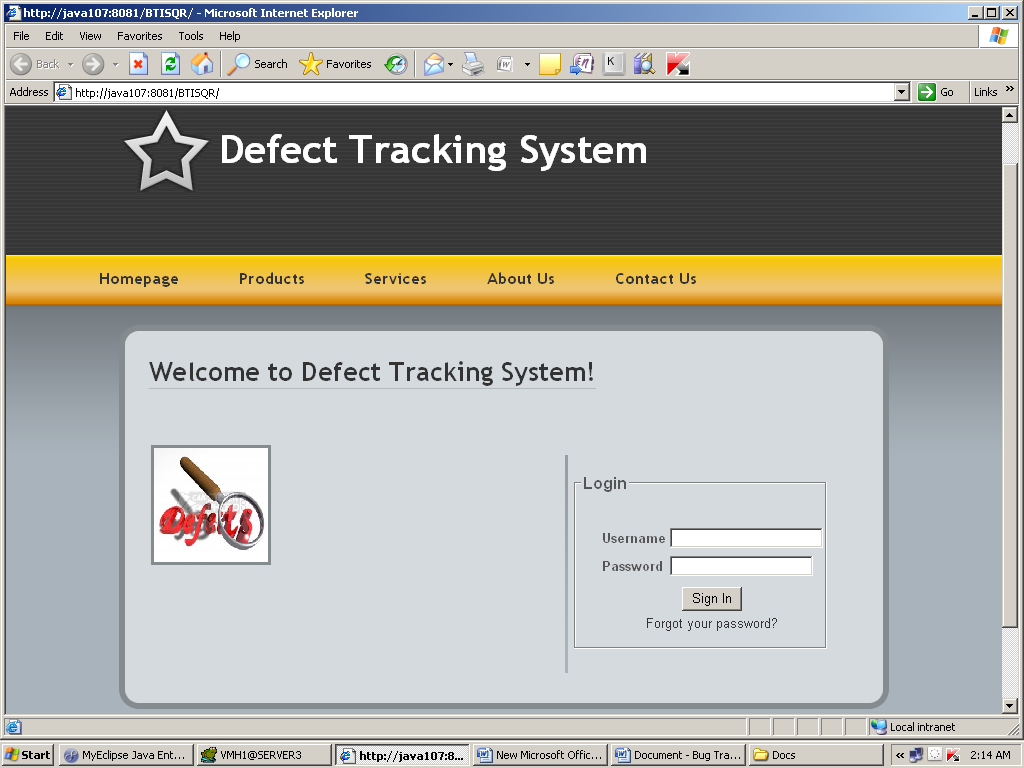
* Must have negative perception.
* Invalid inputs must be used for test.

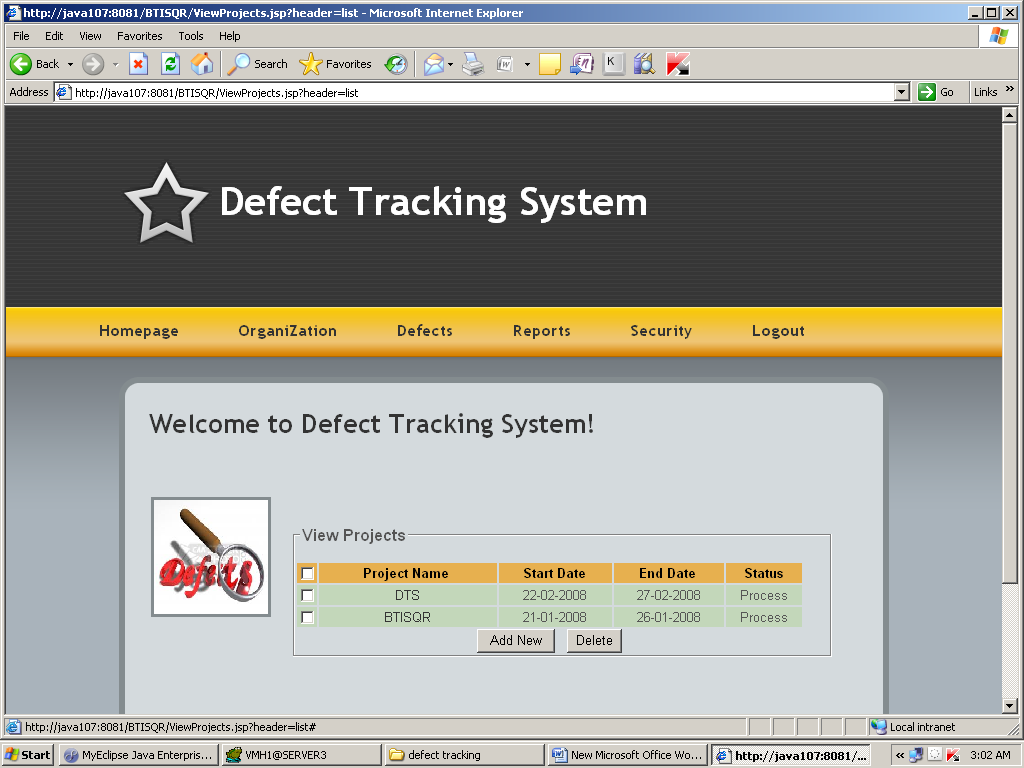
***Example for Negative Test cases***:

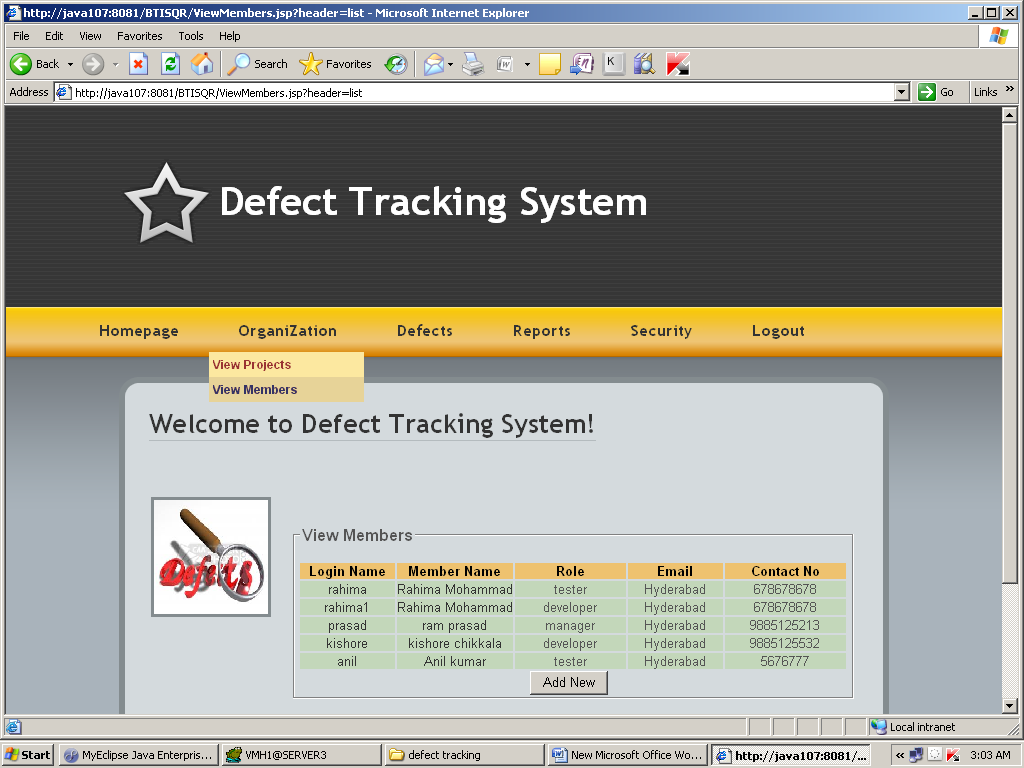
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T.C.No | Description | Expected value | Actual value | Result |
| 1 | Try to modify The information in date and time | Modification should not be allow |  |  |
| 2 | Enter invalid data in to the student details form, click on save | It should not accept invalid data, save should not allow |  |  |

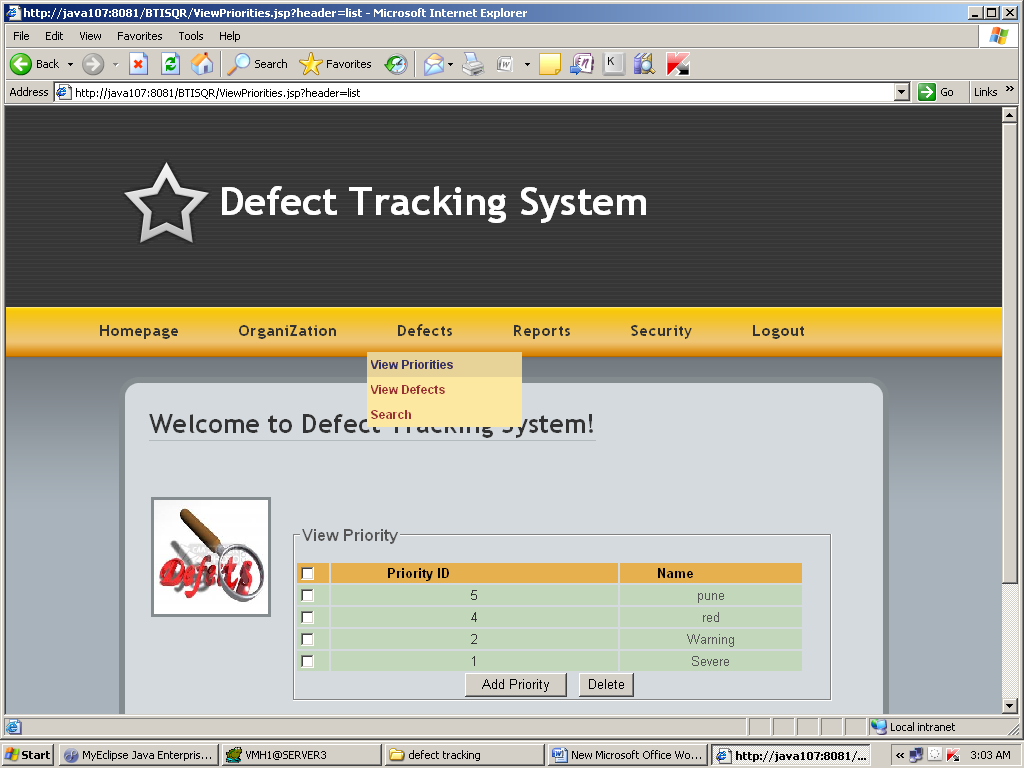
**3.7 Snap shorts :**

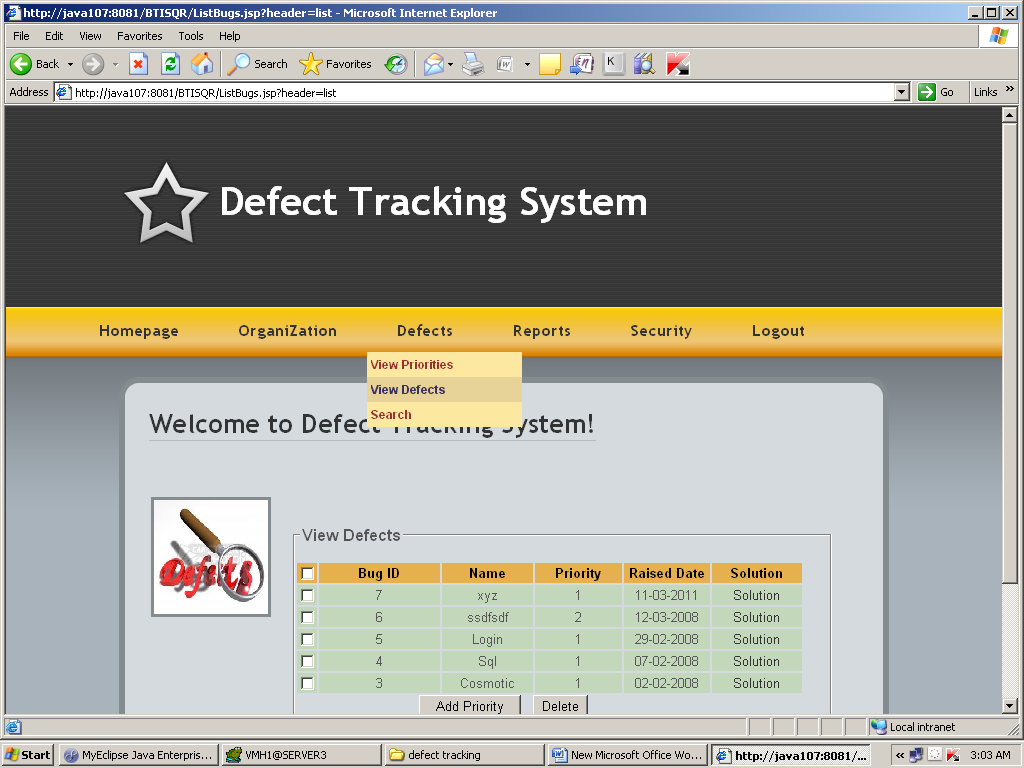




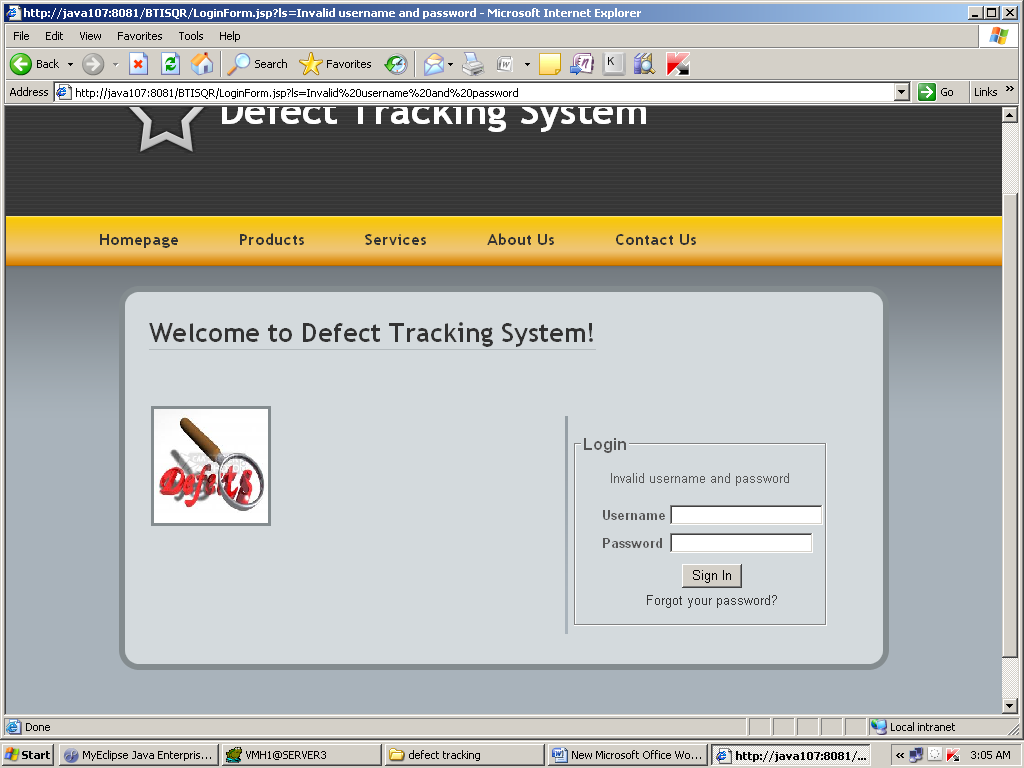
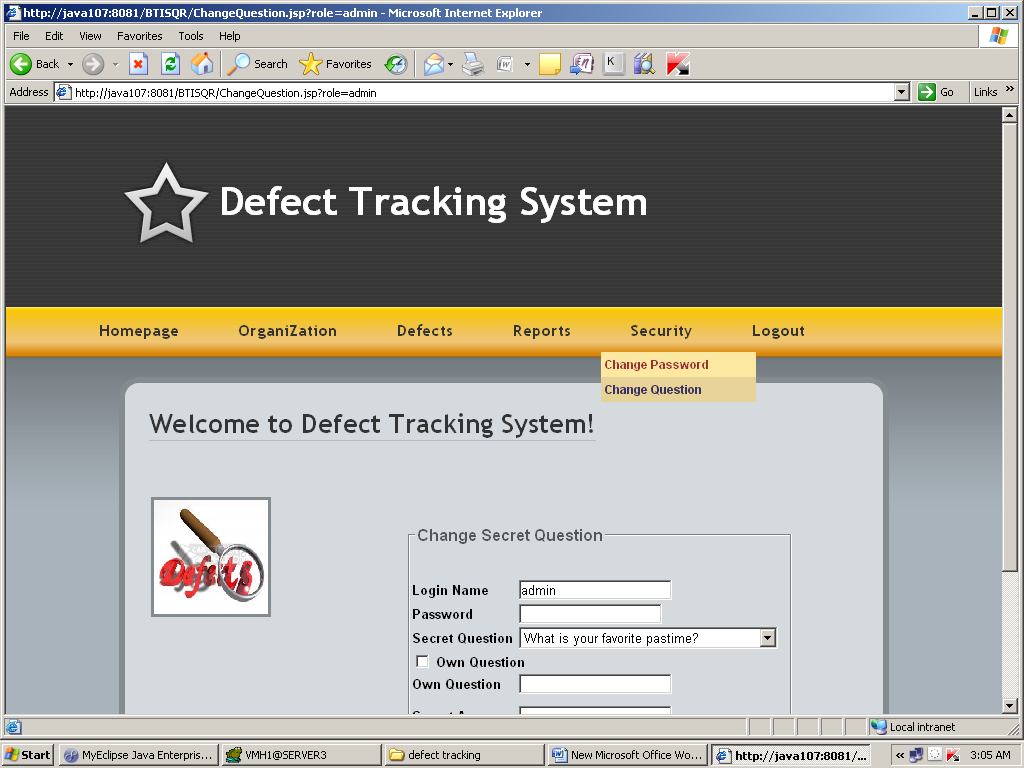
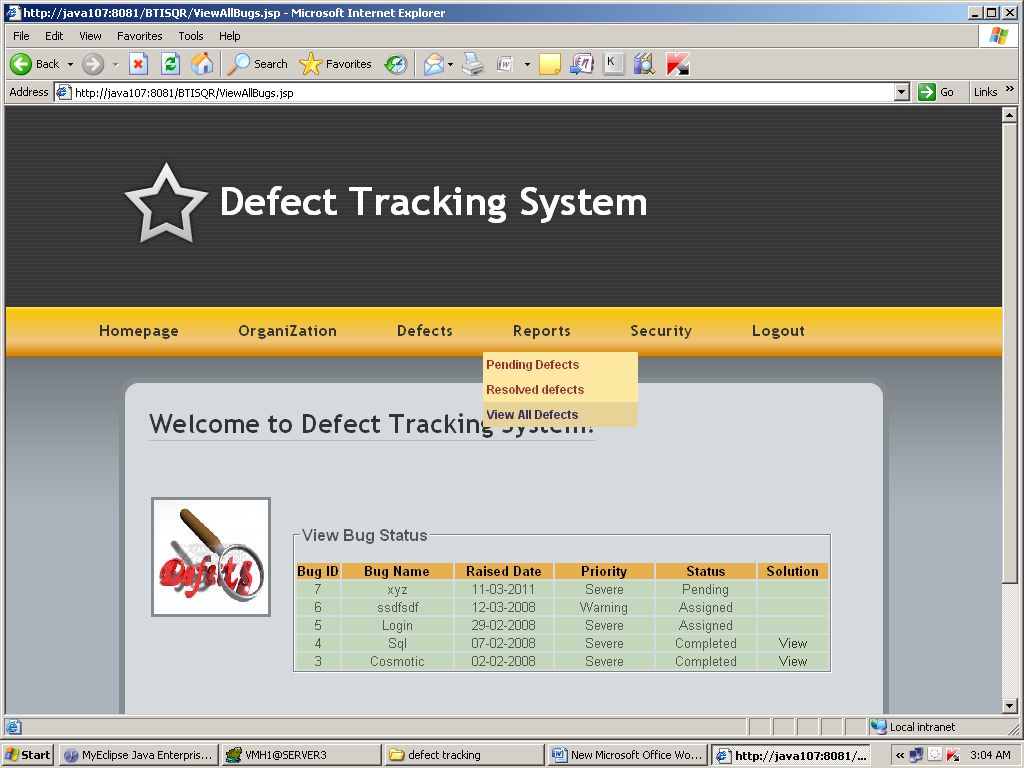
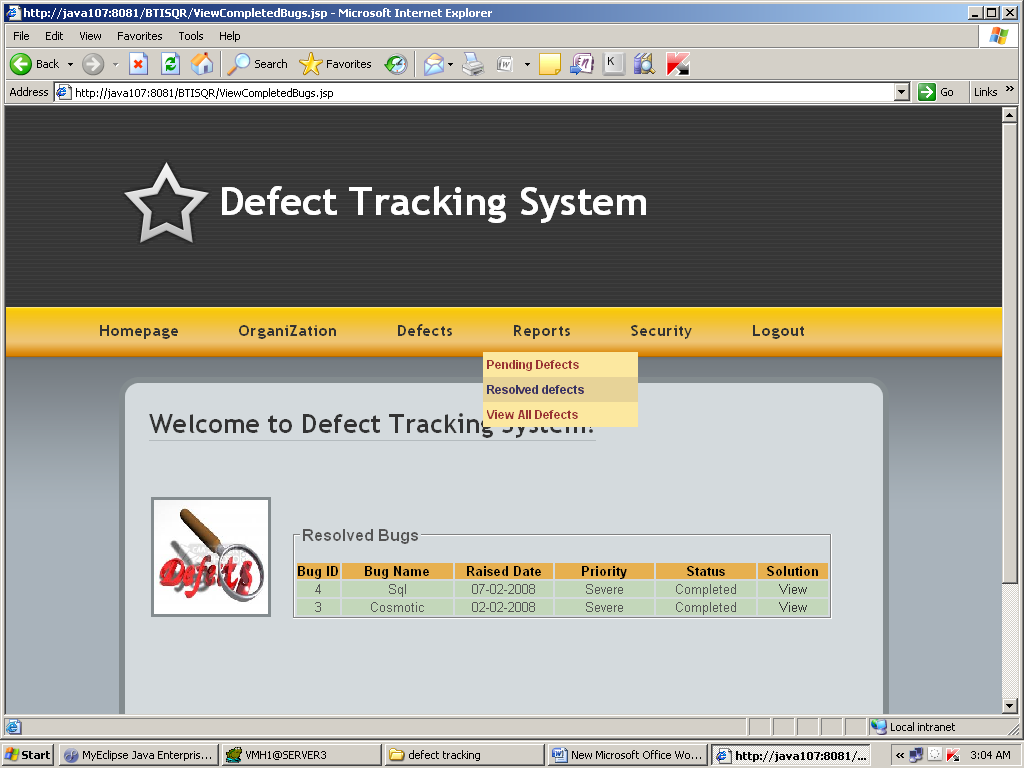
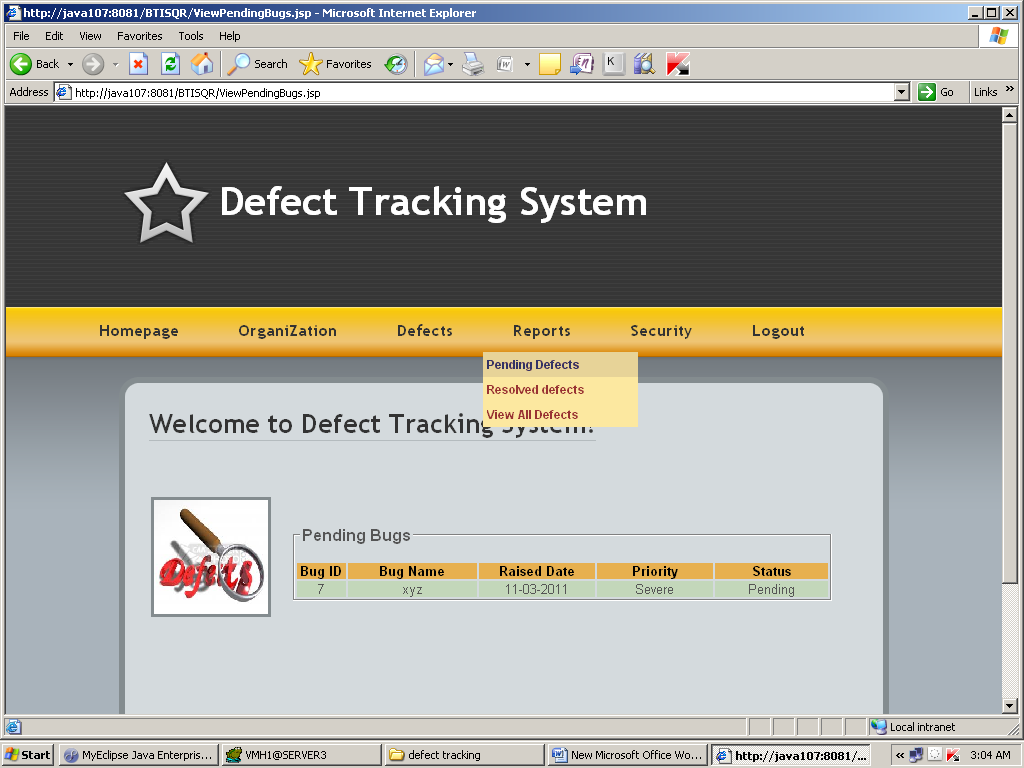








**3.8 Reports :**



**4. Conclusion :**

Defect Tracking System is very useful for removing defects from project module. Only if features mentioned in document are extended. In future users may also be possibly notifying the name/Id of defect creator so that organization can prevent their systems. The project is identified by the merits of the system offered to the user. The merits of this project are as follows: -

4.1 ADVANTAGES:

(i) It’s a web-enabled scheme.

(ii) This task offers user to enter the data through

simple and interactive forms. This is very useful for the client to enter the desired information through so much simplicity.

(iii) The user is mainly more worried about the validity of the data, whatever he is entering. There are checkson every stages of any new creation, data entry or update so that the user cannot enter the invalid data, which can build problems at later date.

(iv) Sometimes the user finds bug in the later stages of using Project that he needs to update some of the

Information that he entered earlier. There are options

for him by which he can update the records. Moreover there is limitation for his that he cannot change the primary data field. This keeps the validity of the data to longer extent.

Current Defect tracking systems do not effectively elicit all of the information needed by developers. Without this information developers cannot resolve defects in atimely fashion and so we consider that improvements to the way defect tracking systems collect information are essential. This is likely to speed up the process of resolving bugs. In the future, I will move from the current prototype of the interactive system to a full-scale system that can deal with a variety of information to gather.

**4.2 Limitations and Scope for Future Enhancements**

Limitations of the system:

* Only the permanent employees can access the system.
* System works in all platforms and its compatible environments.
* Advanced techniques are not used to check the authorization.

**4.3 Future Enhancements:**

It is not possible to develop a system that makes all the requirements of the \_User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system-

• As the technology emerges, it is possible to upgrade the system and can be adaptable to desired environment.

• Because it is based on object-oriented design, any further changes can be easily adaptable.

• Based on the future security issues, security can be improved using emerging technologies.

• Attendance module can be added • sub admin module can be added

1. **Biblography :**

**5.1 References :**

|  |  |
| --- | --- |
| Core Java™ 2 Volume I – Fundamentals 7th Edition | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
| Core Java™ 2 Volume II – Advanced | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
| Head First Servlets & JSP | * Eric Freeman |
| O’Reilly – SPD | Elisabeth Freeman |
|  |  |
| The Book of JavaScript 2nd Edition | * Thau |
| SPD |  |
|  |  |
| Effective Java – Programming Language Guide | * Joshua Bloch |
| Pearson Education – Sun Microsystems |  |
|  |  |
| Java Database Best Practices | * George Reese |
| O’Reilly – SPD |  |
|  |  |
| JBoss – A Developers Notebook | * Norman Richards |
| O’Reilly – SPD | Sam Griffith |