ICE-3204_NBWebApp

Code Review Report

Version <2.0>

Team Name:

Team Members

Team Leader:

Tazqia Mehrub (16511041)

Code Review:

Kazi Tahmid Rashad (16511016), Md.Tanjil Mostafa Rubel (16511026)

Documentation:

Md.Ismail Bhuiyan (16511023), Tanveer Ahmed Siddiqui (16511037)

05 August, 2018

1. Introduction

1.1 What is Source Code Review?

Source code review is a phase of the software development process where an assessment of existing or developing computer source code is done. It is intended to find mistakes overlooked in software development to eliminate vulnerabilities and to avoid security defects, improving the overall quality of software. This code review is done by the Quality Assurance team to avoid future expensive process handling after delivery.

The source code is normally reviewed by another programmer who inspects the code for mistakes, irregular formatting, or inconsistencies with the System Requirements that may create larger issues with software integration.

1.2 Benefits of Source Code Review

There are many benefits of the source code review. These are shown in points below:

- Improves code quality
- Helps being familiar with the project
- Fixes vulnerabilities **in** initial stage
- Ensures Application complies with set standards
- Removes many security issues from the code
- Cost benefit as the effort required is lessened
- leads to small, self-contained increments
- ensures that ideas can be understood from code
- leads to review discussions visible to everybody
- leaves room to develop alternative solutions
- Removes irregular formatting or inconsistencies
- Ensures to follow the standards defined by the organization

1.3 Purpose of this report

Code Review is a specialized task with the goal of identifying types of weaknesses that exist within a given code base. The task involves both manual and automated review of the underlying source code and identifies specific issues that may be representative of broader classes of weakness inherent in the code. A Secure Code Review does not attempt to identify every issue in the code, but instead attempts to identify types of risk within the code such that mitigation strategies can be devised. The purpose of this document is to present a detailed description of the code review of "ICE-3204_NBWebApp" Java Web Application and gathering all the information necessary to control the program. It will explain the purpose, features, documentation and procedures of the code review. This document is intended for the programmers for further use.

2. Review Summary

The code review of the Web application “ICE-3204_NBWebApp” in Java was completed on August 5, 2018 by a review team consisting of 5 members. The review was performed on code by Tazqia Mehrub, Md.Tanjil Mostafa Rubel and Kazi Tahmid Rashad. The Documentation of the reviewed code was performed by Ismail Bhuiyan and Tanveer Ahmed Siddiqui.
The reviewed code was obtained from Tazqia Mehrub via email attachment on August 04, 2018, and bundled under the file named “ICE-3204_NBWebApp_v2.rar”.
A meeting between the review team was held on July 15, 2018, July 22, 2018 and July 29, 2018 at which information about the code structure was presented along with high level overviews of how things like authentication, data validation were implemented in the code. This information was used by the review team to formulate a plan for the impending review.
The actual review involved a manual investigation of the Java code. Specific source files were not assigned to individual members; rather, each member of the review team attempted to review the entire application. Each reviewer recorded their specific findings within a spreadsheet and assigned risk levels as they felt appropriate. At the end of the review, the team looked across the individual spreadsheets to compare common findings and to perform group reviews of the uncommon findings. The specific findings are presented in the next section.

3. Findings

3.1 Findings after reviewing the code

This section provides a summary of the findings resulting from this review.

There were These code bugs are divided into three categories. Such as-High, Medium, Low according to their risk levels.

In this program there are three classes:

1. bk.java

2. log.java

3. show.java

Automated static code analyzer ‘FindBugs’ has been used for inspecting the code given. Along with that some manual inspection was also done by the code review team. Both the automated and manual inspection resulted in total 13 findings along with duplication.

The following points were kept in mind while reviewing and finding bugs:

1. Documentation problems
2. Error handling
3. Coding conventions
4. Code Modularity
5. Understandability

6. Completeness
7. Code design

3.2 Graphical expression of the risks:

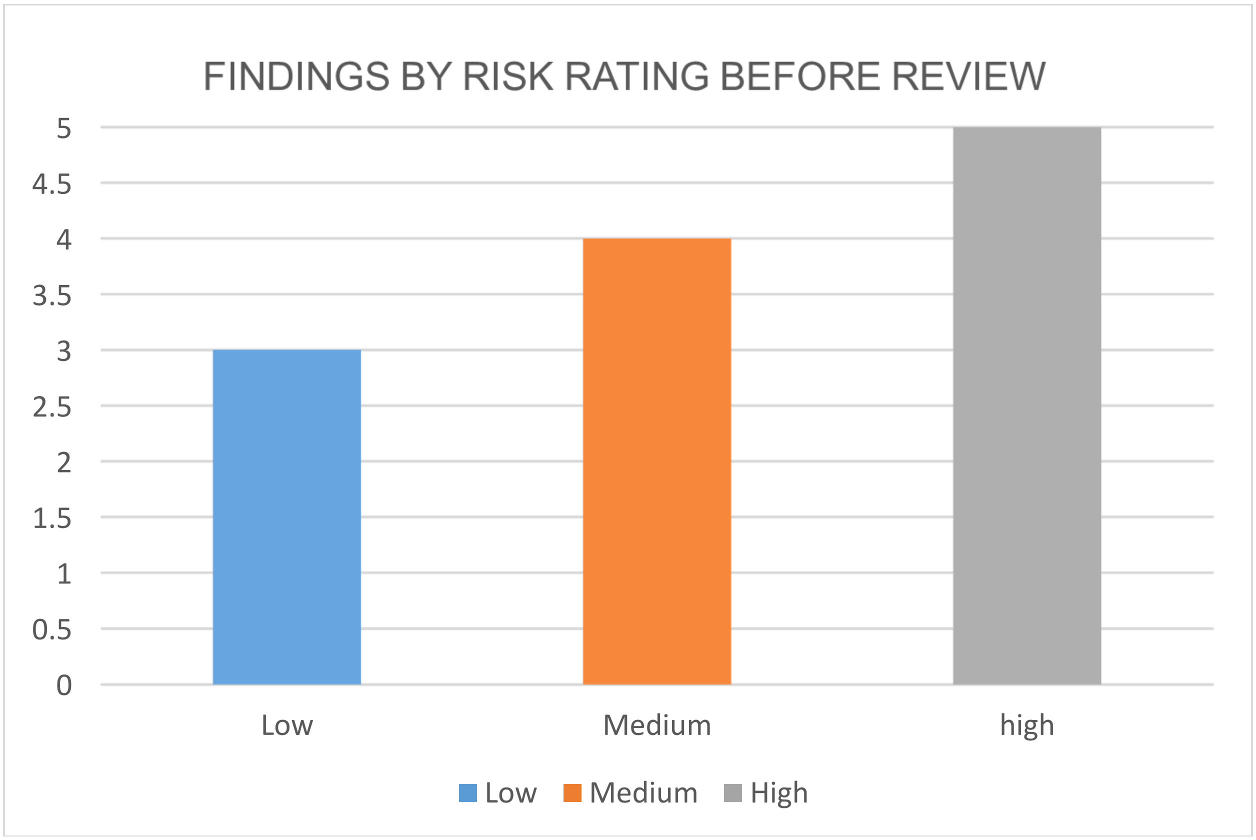


Fig.: 3.1

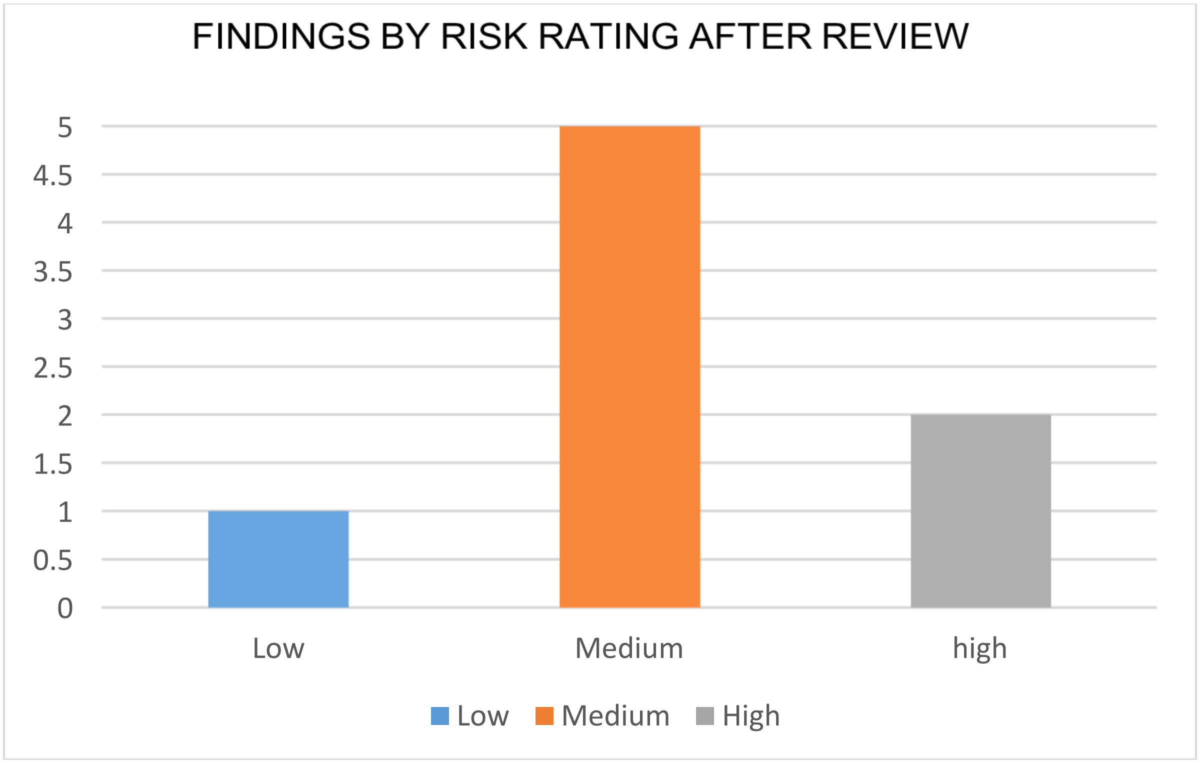

FINDINGS BY RISK RATING AFTER REVIEW
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Low
Medium
high
Low
Medium
High

Fig.: 3.2

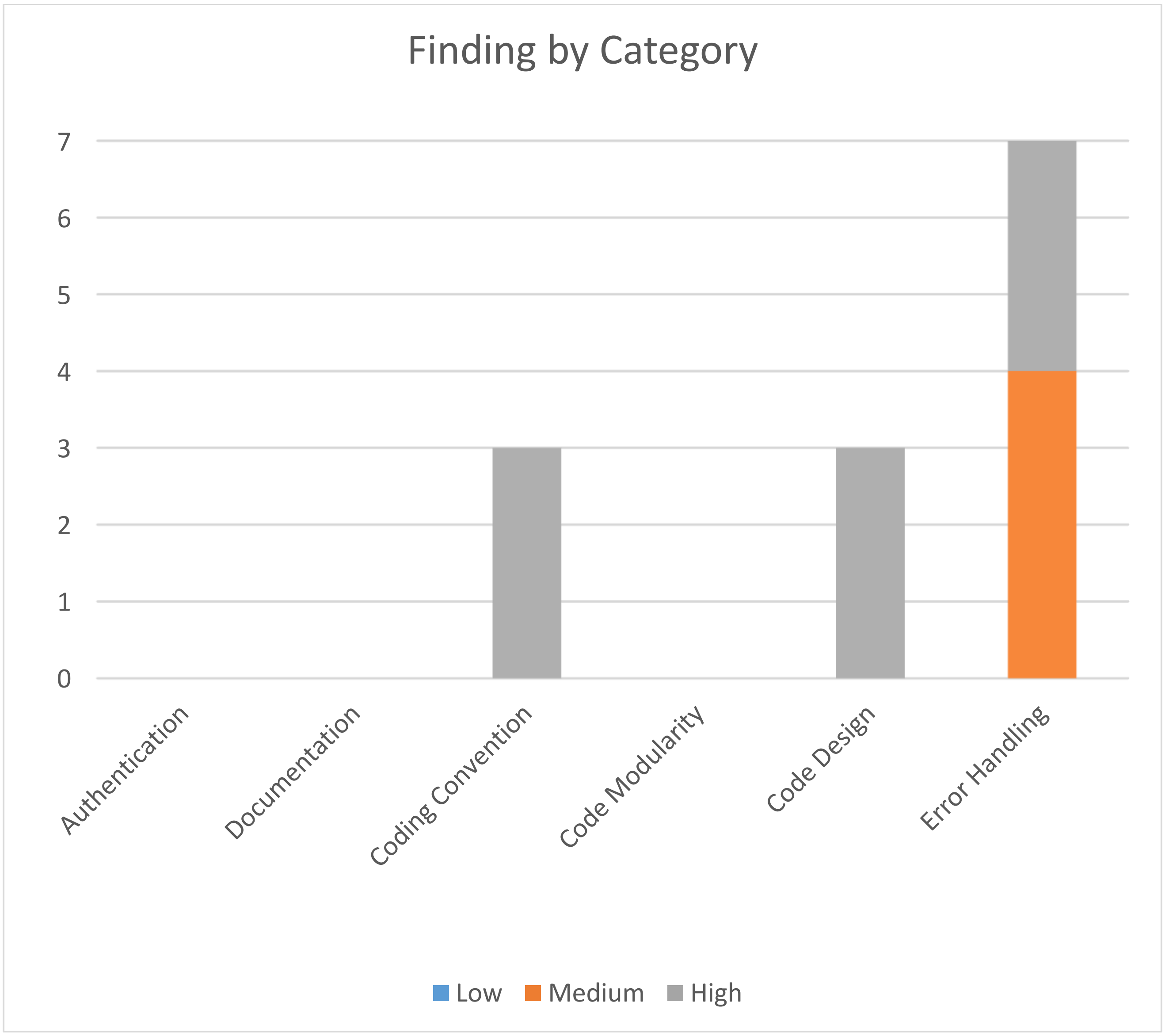

Finding by Category
7
6
5
4
3
2
1
0
Authentication
Documentation
Coding Convention
Code Modularity
Code Design
Error Handling
Low
Medium
High

Fig.: 3.3

4. Finding Details

The detailed description of the bugs have been shown in the table before the code review

Source File	**Line Number**	**Description Risk**	**Risk Level**
src\java\Model2\Controller \Model2Servlet.java	18	This class implements the Serializable interface, but does not define a serialVersionUID field.	Medium
src\java\ShoppingCart\bk.java	20	Class names should start with an upper case letter. Class names should be nouns, in mixed case with the first letter of each internal word capitalized.	Low
src\java\ShoppingCart\bk.java	20	This class implements the Serializable interface, but does not define a serialVersionUID field.	Medium
src\java\ShoppingCart\log.java	22	Coding convention	Low
src\java\ShoppingCart\log.java	22	Class is Serializable, but doesn't define serialVersionUID.This class implements the Serializable interface, but does not define a serialVersionUID field.	
src\java\ShoppingCart\log.java	106	Dead store to local variable	Medium
src\java\ShoppingCart\log.java	113	HTTP cookie formed from untrusted input This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to an HTTP response, it will allow a HTTP response splitting vulnerability.	High

src\java\ShoppingCart\log.java	114	HTTP cookie formed from untrusted input This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to an HTTP response, it will allow a HTTP response splitting vulnerability	High
src\java\ShoppingCart\ show.java	20	Coding convention	Low
src\java\ShoppingCart\ show.java	20	Class is Serializable, but doesn't define serialVersionUID This class implements the Serializable interface, but does not define a serialVersionUID field	Medium
src\java\ShoppingCart\ show.java	31-65	Html code of the view page is written in this section. Presentation layer is merged with business logic which makes the code complex and it is not reusable.	High
src\java\ShoppingCart\ log.java	24-62	Html code of the view page is written in this section. Presentation layer is merged with business logic which makes the code complex and it is not reusable.	High
src\java\ShoppingCart\ bk.java	28-54	Html code of the view page is written in this section. Presentation layer is merged with business logic which makes the code complex and it is not reusable.	High

Table no.: 4.1

The detailed description of the bugs have been shown in the table after the code review and inspection using FindBugs:

Source File	**Line Number**	**Description Risk**	**Risk Level**
src\main\java\org\Model2Servlet.java	18	This class implements the Serializable interface, but does not define a serialVersionUID field.	Medium
src\main\java\org\BookServlet.java	21	This class implements the Serializable interface, but does not define a serialVersionUID field.	Medium
src\main\java\org\BookServlet.java	35	This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.	Low
src\main\java\org\Log.java	25	This class implements the Serializable interface, but does not define a serialVersionUID field.	Medium
src\main\java\org\Log.java	118	This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.	Medium
src\main\java\org\Log.java	125	This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to	High

		an HTTP response, it will allow a HTTP response splitting vulnerability.	
src\main\java\org\Log.java	126	This code constructs an HTTP Cookie using an untrusted HTTP parameter. If this cookie is added to an HTTP response, it will allow a HTTP response splitting vulnerability.	High
src\main\java\org\Log.java	170	This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.	Medium

Table no.: 4.2

5. Code Improvement after Review

The code has been improved after finding all the bugs through both manual and automated process. All the errors have been categorized in different types and have been corrected. Description about the bugs are given here with the improvement process:

5.1 Coding conventions

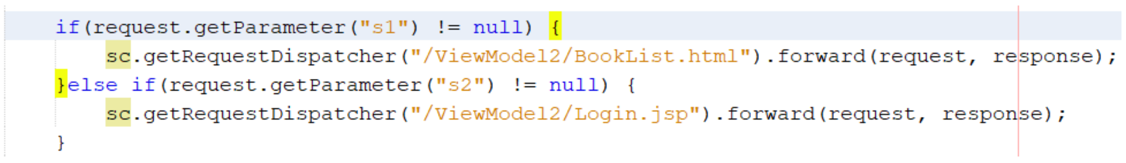
The original source code had problems with the coding conventions. The coding conventions were not followed in many places. In a Java program, class name should always start with uppercase. There were several classes where the class name started with lowercase. The parts where the coding conventions were not followed are:

- bk class under bk.java
- log class under log.java
- show class under show.java

The class names have been changed according to the coding convention.

5.2 Code Style and indentations

A particular code style and indentation have been followed throughout the project. We have followed the 1TBS (OTBS) style throughout the project. Advantages of this style are that the starting brace needs no extra line alone; and the ending brace lines up with the statement it conceptually belongs to. One cost of this style is that the ending brace of a block needs a full line alone.



```
if(request.getParameter("s1") != null) {
    sc.getRequestDispatcher("/ViewModel2/BookList.html").forward(request, response);
}else if(request.getParameter("s2") != null) {
    sc.getRequestDispatcher("/ViewModel2/Login.jsp").forward(request, response);
}
```

5.3 Comments and documentations

After correction and improvement of all the bugs and errors, the documentation of the reviewed code has been done. Necessary comments have also been added to the code for easy understanding.

5.4 Code Cleaning:

All the unnecessary codes have been removed from the project. Under the Source Package, there was a servlet named "Model2Servlet.java" which was totally irrelevant in the code and it has not been used anywhere of the code. That is why, Model2Servlet.java has been removed.

The unused Import Packages have also been removed from the code. As the pages index.html and welcome.jsp were unused, they have also been removed. This was the procedure of code cleaning by removing irrelevant codes.

5.5 Organizing code into framework

In the original code no framework was followed. All the codes were written under java class. Even the viewer side (user interface) was written inside the servlet. The UI and server side codes were mixed together which is not a good practice. The code has been organized in such a way so that the codes are not mixed and can be differentiable.

After review, a defined structure is followed. The framework MVC is used to separate the business logic from presentation layer. A folder named view is created all the user interface is created and the ShoppingCart controller package holds the business logic. The viewer side (UI) codes have been added separately under ViewModel2 folder and the server side codes are under ShoppingCart folder under Source Packages.

The newly created pages are:

- Books. html

- Login. jsp
- Shop. jsp

5.6 Readability of the code

The code has been made readable by using the class name, method name, variable name depending on the functionality of the classes, methods or variables.

For example, on the viewer side the page names Books, Shop, Login have been used. It is easy to guess what will be inside those pages seeing their names. Inside 'Books.html' there is a list of books from where books can be chosen. Inside 'Shop.html' the selected book names will be shown. Inside 'Login' there will be the login page for the buyers.

Similarly, this has been followed in the server side. Hence, making the code readable and understandable.

6. Conclusion

Code reviews are one of the many methods to verify and improve the code. The reviewer has to get accustomed with the source code structure before the review can even start. Static analysis tools can be used to improve the efficiency and the coverage for large code bases, but manual verification of the results is always necessary. In addition, logical flaws cannot be detected by static analysis tools, since identifying these types of flaws requires understanding of the business process that the software supports.

In this project, most of the findings were language-specific. The main focus was on the code structure of a Java Web Application as it is very important from the development point of view.

The final conclusion can be stated that the coding conventions have been followed, bugs have been removed through automated and manual approach. There were few bugs which could not be fixed and those are not of high severity. Also those findings cannot be considered as directly security flaws or development flaws.