Concordia University
Engineering and Computer Science

SOEN 6431 - Software Comprehension And Maintenance

Deliverable 2 - Reengineering Opportunity

DÉJÀ VU

Submitted to : Dr. Pankaj Kamthan

Summarized by:
A K M Saifun Nabi(40222168)
Dhruviben Jigeshkumar Modi(40166396)
Sumit Vasharambhai Monapara(40197174)
Harshal Jigeshkumar Modi (40195060)

Github Repository Link

Contents

1. Abstract

Continuous evolution is essential for a software system's long-term market success. As part of this evolution, the original software must be continuously updated and improved in order to meet the shifting needs of the market. Software maintenance is the term used to describe this ongoing improvement process. It is crucial to carefully examine the source code and documentation and have a deep understanding of the system in order to carry out software maintenance and apply beneficial modifications.

We have chosen the R system called Billing System from the 4 candidate systems which we described in deliverable 1. In this deliverable 2, we make changes in source code R as we mentioned in deliverable 1. In this assignment, we make a report about the change in relevant software metrics before and after the re-engineering.

2. Introduction

Software re-engineering offers numerous advantages such as improved maintainability, enhanced performance, and increased flexibility. Debugging, reviewing, and testing of source code play critical roles in ensuring software reliability, identifying and resolving defects, and enhancing overall code quality. These practices contribute to the development of robust and stable software systems, leading to better user experiences and increased customer satisfaction.

The Billing System is a comprehensive store billing application developed in Java with a user-friendly graphical user interface (GUI). The system generates invoices tailored to each state's requirements. The main file, AdminPanel.java, facilitates various CRUD operations such as adding, deleting, and updating products. Additional Java files like updateProduct.java, deleteCashier.java, deleteProduct.java, showStocks.java, and addCashier.java handle specific tasks related to managing products, cashiers, and displaying stock information. Together, these files create an efficient and versatile billing system for stores.

Choosing a Java billing system project based on the team's familiarity with the language provides several benefits, including enhanced collaboration, reduced learning curve, and improved productivity. Implementing the project with the MVC design pattern ensures a proper coding structure, modularity, and maintainability. The effective utilization of Java's exception handling mechanism further enhances the system's reliability and stability. To ensure efficient operations, seamless integration with databases is necessary. This integration facilitates the storage, retrieval, and manipulation of critical data, allowing accurate billing processes, streamlined reporting, and comprehensive analysis. Proper database integration ensures data consistency, security, and scalability. This System has utilized JDBC API. The JDBC API provides a standardized set of interfaces, allowing Java applications to establish connections with various databases. It simplifies database interactions by providing methods to execute SQL queries, retrieve data, and manage transactions.

3. List of Source Code Undesirables

Our team has extensively utilized the SonarQube tool to diligently scrutinize our deliverable and detect any undesired elements present in the codebase. The findings we encountered cover a wide spectrum of undesirables, including critical issues such as use of the deprecated libraries, potential security vulnerabilities, and architectural weaknesses, as well as less critical matters such as coding style inconsistencies and suboptimal code blocks.

In this section, we present a detailed breakdown of each identified undesirable, providing comprehensive information such as its type, category, specific code smell type, and a comprehensive description. We aim to offer a profound understanding of each issue to facilitate effective resolution and enhance the overall quality of our deliverable. Throughout this meticulous process, we have meticulously documented a summary for each code smell, providing an overview of its impact on the codebase and potential implications. In total, we have identified and resolved a total of 20 findings, signifying our unwavering commitment to producing a high-quality deliverable that adheres to industry best practices and standards.

We have categorized three types of undesirables with their respective explanations: Bugs, Vulnerabilities, and Code Smells.

Bug: A bug refers to an error, flaw, or unintended behavior in the software code that leads to incorrect or unexpected results during execution.

Vulnerability: A vulnerability is a weakness in the software system that could be exploited by an attacker to compromise its security, integrity, or availability.

Code Smell: Code smell indicates a piece of code that is poorly designed or written, making it difficult to maintain, understand, or extend. It doesn't necessarily lead to bugs but hinders code quality and may increase the likelihood of introducing defects.

1	Type	Bug
	Severity	Blocker
	Observation	Use try-with-resources or close this ”Statement” in a ”finally” clause.
	Code Smell Type	Resource Management - Finally block clean-up
	Code Smell Summary	Resources like connections, streams, and files that implement the Closeable or AutoCloseable interface should be closed after use to release system resources properly. Preferably, when a class implements AutoCloseable, the ”try-with-resources” pattern should be used to ensure automatic resource cleanup
	Frequency	16

4. Re-engineering Methods for Undesrirables

2	Type	Bug
	Severity	Major
	Observation	A ”NullPointerException” could be thrown; ”conn” is nullable here.
	Code Smell Type	Implementation Smells - Null Pointer Dereference.
	Code Smell Summary	Attempting to dereference or access members/methods of a reference that is null will cause a ”NullPointerException,” leading to abrupt program termination.We should always perform null checks on nullable references before accessing them.
	Frequency	15

3	Type	Vulnerability
	Severity	Blocker
	Observation	Add password protection to this database.
	Code Smell Type	Inadequate Security Measures
	Code Smell Summary	When relying on the password authentication mode for the database connection, a secure password should be chosen. This rule raises an issue when an empty password is used.
	Frequency	1

4	Type	Code Smell
	Severity	Critical
	Observation	Use static access with **javax.swing.WindowConstants** for DISPOSE_ON_CLOSE
	Code Smell Type	Static Overuse: Excessive Static Access.
	Code Smell Summary	In the interest of code clarity, static members of a base class should never be accessed using a derived type’s name. Doing so is confusing and could create the illusion that two different static members exist.
	Frequency	2

5	Type	Code Smell
	Severity	Critical
	Observation	Define a constant instead of duplicating this literal **”XXXX”** N times.
	Code Smell Type	Duplicated Variable - Magic String
	Code Smell Summary	Duplicated string literals make the process of refactoring error-prone since you must be sure to update all occurrences, but with assigning to constant, we need to update only once and access anywhere.
	Frequency	25

6	Type	Code Smell
	Severity	Critical
	Observation	Add a nested comment explaining why this method is empty or throw Exception
	Code Smell Type	Configuration Smells : Empty method
	Code Smell Summary	There are several reasons for a method not to have a method body: if the method will not be used in the future, throw an UnSupportedOperationException or if overriding it, add a nested comment explaining why this method is empty.
	Frequency	1

7	Type	Code Smell
	Severity	Critical
	Observation	Make **”XXXX”** transient or serializable.
	Code Smell Type	Implementation Smells: Serialization Smell
	Code Smell Summary	Fields in a Serializable class must themselves be either Serializable or transient,even if the class is never explicitly serialized or deserialized. This rule raises an issue on non-Serializable fields.
	Frequency	1

8	Type	Code Smell
	Severity	Critical
	Observation	Refactor this method to reduce its Cognitive Complexity from 23 to the 15 allowed.
	Code Smell Type	Complex code: High Cognitive Complexity
	Code Smell Summary	Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain.
	Frequency	2

9	Type	Code Smell
	Severity	Critical
	Observation	Use static access with **javax.swing.WindowConstants** for ”EXIT_ON_CLOSE”
	Code Smell Type	Static Overuse: Excessive Static Access
	Code Smell Summary	In the interest of code clarity, static members of a base class should never be accessed using a derived type’s name. Doing so is confusing and could create the illusion that two different static members exist.
	Frequency	1

10	Type	Code Smell
	Severity	Critical
	Observation	Add a default case to this switch.
	Code Smell Type	Switch statements
	Code Smell Summary	The requirement for a final default clause is defensive programming. The clause should either take appropriate action, or contain a suitable comment as to why no action is taken.
	Frequency	3

11	Type	Code Smell
	Severity	Major
	Observation	Replace this use of **System.out \System.err** by a **logger**.
	Code Smell Type	Implementation Smells: Logging
	Code Smell Summary	Everyone must follow the logging patterns to easily retrieve the logs for all users, the format of the log should be uniform across applications, and logged data must be recorded for future use.
	Frequency	3

12	Type	Code Smell
	Severity	Major
	Observation	Make this anonymous inner class a lambda.
	Code Smell Type	Anonymous Class Smell
	Code Smell Summary	Before Java 8, the only way to partially support closures in Java was by using anonymous inner classes, but the syntax was unclear and unwieldy, so after Java 8, anonymous inner classes should be replaced by lambdas to highly increase the readability.
	Frequency	22

13	Type	Code Smell
	Severity	Major
	Observation	This block of commented-out lines of code should be removed.
	Code Smell Type	Implementation Smells - Comments
	Code Smell Summary	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required
	Frequency	7

14	Type	Code Smell
	Severity	Major
	Observation	Remove this useless assignment to local variable **”XXXX”**.
	Code Smell Type	Resource Management - Dead Store
	Code Smell Summary	dead store happens when a local variable is assigned a value that is not read by any subsequent instruction and leads to waste of storage, Therefore all calculated values should be used.
	Frequency	4

15	Type	Code Smell
	Severity	Minor
	Observation	Rename this class name to match the regular expression ’[̂A-Z][a-zA-Z0-9]*$’..
	Code Smell Type	Configuration Smells : Inconsistent Naming Convention
	Code Smell Summary	Shared coding conventions allow teams to collaborate effectively. This rule allows us to check that all class names match a provided regular expression..
	Frequency	9

16	Type	Code Smell
	Severity	Minor
	Observation	Remove this unused import **”AB.CD.EF”**.
	Code Smell Type	Architectural Smells - Unused Packages
	Code Smell Summary	The import part of a file should be handled by the Integrated Development Environment (IDE), not manually by the developer. Leaving them in reduces the code’s readability since their presence can be confusing.
	Frequency	28

17	Type	Code Smell
	Severity	Info
	Observation	Complete the task associated to this TODO comment..
	Code Smell Type	Implementation Smells : Comments
	Code Smell Summary	**TODO** tags are commonly used to mark places where some more code is required, but which the developer wants to implement later. This rule is meant to track those tags and to ensure that they do not go unnoticed.
	Frequency	26