Graphical user interface, text, application

Description automatically generated

[**SOEN 6431**](https://github.com/manimayan/SOEN_6431_Deja_Vu) **- Software Comprehension and Maintenance**

**Summer 2022**

**Deliverable 2-Reengineering Operationalization**

[GitHub link](https://github.com/manimayan/SOEN_6431_Deja_Vu.git)

***Authors***

Manimaran Palani

Iphigenia Pappas

Heet Patel

Kevinkumar Patel

Venis Patel

https://www.overleaf.com/project/6285ba87cc2d8b26da21563d

Table of Contents

[**1.** **Introduction** 3](#_Toc109325998)

[**2.** **Source Code Undesirables Summary** 3](#_Toc109325999)

[**3.** **Re-Engineering Methods for Undesirables** 3](#_Toc109326000)

[**4.** **Locations of the Source Code Undesirables** 3](#_Toc109326001)

[**5.** **Software Metric Log** 3](#_Toc109326002)

[**6.** **Refactoring Report** 3](#_Toc109326003)

[**7.** **Software Specifications** 3](#_Toc109326004)

[**8.** **Refactored Source Code of R** 3](#_Toc109326005)

[**9.** **References** 3](#_Toc109326006)

# **Introduction**

For a software system to be able to successfully sustain in the market, constant evolution becomes of utmost importance. Evolution refers to continuous developments and updates in the originally created software system to match the ever-changing market demands at any given point of time. This process of refining the software to keep it abreast of the trends and demands is termed as Software Maintenance. To carry out a successful maintenance by implementing positive changes in the software systems, the analysis of source code and documentation and having a comprehensive understanding of the system holds the most importance. For this assignment, we choose a candidate R from five different repositories chosen by respective team members. We dive into the detailed analysis of the ‘undesirables’ in the repositories using TeamScale before tapering down to the candidate R. We further explain the repudiation rationale for the same. The chosen candidate R provides us with the opportunity to reengineer the software program by rectifying the ‘undesirables’ and thus improving its efficiency and maintainability.

# **Source Code Undesirables Summary**

|  |  |  |
| --- | --- | --- |
| **1** | **Findings** | Clone with 2 instances of length 16 |
| **Occurrences** | 4 |
| **Type** | Redundancy/Clones |
| **Category** | Code Duplication |
| **Code Smell type** | ***Test Smells*** - Test Code Duplication |
| **Code Smell Summary** | Test code may contain undesirable duplication. In particular the parts that set up test fixtures are susceptible to this problem. |
|  |  |  |
| **2** | **Findings** | Empty block: method |
| **Occurrences** | 2 |
| **Type** | Code Anomalies/cqse-empty-block |
| **Category** | Comprehensibility |
| **Code Smell type** | ***Configuration Smells*** - Unnecessary Abstraction |
| **Code Smell Summary** | A class, 'define', or module must contain declarations or statements specifying the properties of a desired system. An empty class, 'define', or module shows the presence of unnecessary abstraction smell and thus must be removed. |
|  |  |  |
| **3** | **Findings** | Commented Out Code |
| **Occurrences** | 2 |
| **Type** | Comments/Commented Out Code |
| **Category** | Comprehensibility |
| **Code Smell type** | ***Implementation Smells*** - Comments |
| **Code Smell Summary** | This smell occurs when comments are used as deodorant to explain the bad code. |
|  |  |  |
| **4** | **Findings** | Star import of `javax.persistence.\*` should not be used |
| **Occurrences** | 2 |
| **Type** | Code Anomalies/cqse-no-star-imports |
| **Category** | Bad Practice |
| **Code Smell type** | ***Design Smells*** - Obsolete Imports |
| **Code Smell Summary** | This smell occurs when certain classes are no longer used in a software system but loaded due to improper signature of import statements. Classes that are no longer in use will burden the system with obviously obsolete functionality. |
|  |  |  |
| **5** | **Findings** | Unused import: `com.userfront.domain.SavingsTransaction` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-java-unused-imports |
| **Category** | Unused Code |
| **Code Smell type** | ***Architecture Smells*** - Unused Packages |
| **Code Smell Summary** | Packages that are not in use burden the system with clearly obsolete functionality. |
|  |  |  |
| **6** | **Findings** | Method `System.out.println` should not be called |
| **Occurrences** | 3 |
| **Type** | Code Anomalies/cqse-java-unwanted-method-calls |
| **Category** | Discouraged APIs |
| **Code Smell type** | ***Design Smells*** - Poltergeist |
| **Code Smell Summary** | Irrelevant classes: An irrelevant class is a class that does not have any meaningful behaviour in the design. These types of classes are characterised for being composed only of get, set and/or print methods |
|  |  |  |
| **7** | **Findings** | `SimpleDateFormat` constructor should specify `Locale` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-avoid-creating-simple-date-format-without-locale |
| **Category** | API Misuse |
| **Code Smell type** | ***Design Smells*** -Divergent Change |
| **Code Smell Summary** | You find yourself having to change many unrelated methods when you make changes to a class. For example, when adding a new product type, you have to change the methods for finding, displaying, and ordering products. |
|  |  |  |
| **8** | **Findings** | Interface comment missing |
| **Occurrences** | 151 |
| **Type** | Comments/Missing Interface Comment |
| **Category** | Comment Completeness |
| **Code Smell type** | ***Shortage Smells*** - Debt |
| **Code Smell Summary** | Similarly, to Chant that covers up imperfect fragments with comments in natural language, there could be pieces missing entirely from the grammar and replaced with comments. If the comments admit clearly what is missing, use searchable tags like “TODO” or “FIXME” and are intended to use as a backlog. |
|  |  |  |
| **9** | **Findings** | TODO Auto-generated method stub |
| **Occurrences** | 3 |
| **Type** | Comments/Task Tags |
| **Category** | Documentation |
| **Code Smell type** | ***Implementation Smells*** - Comments |
| **Code Smell Summary** | This smell occurs when comments are used as deodorant to explain the bad code. |
|  |  |  |
| **10** | **Findings** | Catch clause catches generic exception `Exception` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-catch-high-level-exception |
| **Category** | Imprecise Handling |
| **Code Smell type** | ***Implementation Smells*** - Catch Block |
| **Code Smell Summary** | This smell occurs when a catch block of an exception is improperly handled with non-matching or non-precise exception libraries. |
|  |  |  |
| **11** | **Findings** | Logger should be specified with `UserServiceImpl.class` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-logger-with-wrong-specified-class |
| **Category** | Logging |
| **Code Smell type** | ***Implementation Smells*** - Attribute name and type are opposite |
| **Code Smell Summary** | The name of an attribute is in contradiction with its type as they contain antonyms. \Example: attribute start of type Association End. The use of antonyms can induce wrong assumptions. |
|  |  |  |
| **12** | **Findings** | Clone with 2 instances of length 28 |
| **Occurrences** | 2 |
| **Type** | Redundancy/Clones |
| **Category** | Code Duplication |
| **Code Smell type** | ***Test Smells*** - Test Code Duplication |
| **Code Smell Summary** | Test code may contain undesirable duplication. In particular the parts that set up test fixtures are susceptible to this problem. |
|  |  |  |
| **13** | **Findings** | Private field `env` is never read. |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-java-avoid-unused-private-fields |
| **Category** | Unused Code |
| **Code Smell type** | ***Design Smells*** - Unutilized Abstraction |
| **Code Smell Summary** | This smell arises when a variable is left unused (either not directly used or not reachable). |
|  |  |  |
| **14** | **Findings** | Empty block: constructor |
| **Occurrences** | 4 |
| **Type** | Code Anomalies/cqse-empty-block |
| **Category** | Comprehensibility |
| **Code Smell type** | ***Configuration Smells*** - Unnecessary Abstraction |
| **Code Smell Summary** | A class, 'define', or module must contain declarations or statements specifying the properties of a desired system. An empty class, 'define', or module shows the presence of unnecessary abstraction smell and thus must be removed. |
|  |  |  |
| **15** | **Findings** | Unused import: `com.userfront.domain.PrimaryTransaction` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-java-unused-imports |
| **Category** | Unused Code |
| **Code Smell type** | ***Architecture Smells*** - Unused Packages |
| **Code Smell Summary** | Packages that are not in use burden the system with clearly obsolete functionality. |
|  |  |  |
| **16** | **Findings** | Name `com.userfront.service.UserServiceImpl` violates naming convention. Should be one of `[a-z][a-z\_0-9.]\*` |
| **Occurrences** | 5 |
| **Type** | Naming/JAVA |
| **Category** | Comprehensibility |
| **Code Smell type** | ***Configuration Smells (Implementation)*** - Inconsistent Naming Convention |
| **Code Smell Summary** | The used naming convention deviates from the recommended naming convention. |
|  |  |  |
| **17** | **Findings** | Remove this use of `getBytes` |
| **Occurrences** | 1 |
| **Type** | Compatibility issues |
| **Category** | Correctness |
| **Code Smell type** | ***Implementation Smells*** - “Get” - more than an accessor |
| **Code Smell Summary** | A getter that performs actions other than returning the corresponding attribute without documenting it.Example: method getImageData which, no matter the attribute value, every time returns a new object. |
|  |  |  |
| **18** | **Findings** | Use `BigDecimal.ZERO` instead of creating new object |
| **Occurrences** | 2 |
| **Type** | Code Anomalies/cqse-unnecessary-big-integer-instantiation |
| **Category** | Performance |
| **Code Smell type** | ***Performance Smells*** - Unnecessary Processing |
| **Code Smell Summary** | This smell occurs when processing is not needed or not needed at that time. Solution: Delete the extra processing steps, reorder steps to detect unnecessary steps earlier, or restructure to delegate those steps to a background task. |
|  |  |  |
| **19** | **Findings** | Comment should not contain `/\*` |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-nested-comment |
| **Category** | Malformed Comments |
| **Code Smell type** | ***Configuration Smells (Implementation)*** - Improper Quote Usage |
| **Code Smell Summary** | Single and double quotes are not used properly. For example, Boolean values should not be quoted, and variable names should not be used in single quoted strings. |
|  |  |  |
| **20** | **Findings** | The `printStackTrace()` method should not be called |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-print-stack-trace |
| **Category** | Imprecise Handling |
| **Code Smell type** | ***Implementation Smells*** - Incomplete Library Class |
| **Code Smell Summary** | The author of the library has not provided the features you need or has refused to implement them. invoking printStackTrace() changes the destination pointed to by System.err by redirecting the process to a file/device whose contents may be ignored by personnel. |
|  |  |  |
| **21** | **Findings** | Throw of generic exception “Exception” |
| **Occurrences** | 1 |
| **Type** | Code Anomalies/cqse-throw-high-level-java-exceptions |
| **Category** | Imprecise Handling |
| **Code Smell type** | ***Implementation Smells*** - Catch Block |
| **Code Smell Summary** | This smell occurs when a catch block of an exception is improperly handled with non-matching or non-precise exception libraries. |

# **Re-Engineering Methods for Undesirables**

# **Locations of the Source Code Undesirables**

# **Software Metric Log**

# **Refactoring Report**

# **Software Specifications**

# **Refactored Source Code of R**

# **References**