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. |

Graphical user interface, application

Description automatically generated

Figure 1: Lines of Code vs. Findings Trend of **Candidate R** - Online Banking System

Chart

Description automatically generated

Figure 2 : Findings Summary of **Candidate R** - Online Banking System

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3: Undesirables Summary of **Candidate R** - Online Banking System

# **Re-Engineering Methods for Undesirables**

|  |  |  |
| --- | --- | --- |
| **1** | **Findings** | Clone with 2 instances of length 16 |
| **Quality Indicator** | ***Cloning*** |
| **Re-Engineering Rules/Standards** | The best strategy for removal of a clone depends on the context. Typical solutions are The extraction of utility methods, Introduction of common base classes, or application of code generators. |
| **Refactored By** | Manimaran Palani |
|  |  |  |
| **2** | **Findings** | Empty block: method |
| **Quality Indicator** | ***Empty blocks*** |
| **Re-Engineering Rules/Standards** | Empty blocks are typically not required and can confuse the reader as it is unclear if the empty block was intended or should be filled later on. To clarify this, developers should place an explanatory comment into the block. C++ destructors: Empty and non-virtual destructors are probably not useful. They should either be deleted or made virtual (if the class is intended for subclassing). |
| **Refactored By** | Manimaran Palani |
|  |  |  |
| **3** | **Findings** | Commented Out Code |
| **Quality Indicator** | ***Commented-out code*** |
| **Re-Engineering Rules/Standards** | Commented-out code makes it hard for a reader to determine whether the code can be deleted or is just commented out temporarily. In the context of version control systems, commented-out code should not be necessary. Detecting commented-out code is performed heuristically, by looking for code-like patterns, such as trailing semicolons or many operators. If your commenting style uses these frequently also for non-code comments, consider disabling this check. |
| **Refactored By** | Manimaran Palani |
|  |  |  |
| **4** | **Findings** | Star import of `javax.persistence.\*` should not be used |
| **Quality Indicator** | ***No star imports*** |
| **Re-Engineering Rules/Standards** | Star imports must not be used, because they clutter the local namespace. |
| **Refactored By** | Iphigenia Pappas |
|  |  |  |
| **5** | **Findings** | Unused import: `com.userfront.domain.SavingsTransaction` |
| **Quality Indicator** | ***Unused/duplicated imports (Java)*** |
| **Re-Engineering Rules/Standards** | Unused/Duplicated imports are not needed and make the code files unnecessarily large. Instead of manually removing them, check if your IDE can do this automatically, e.g. on each save. |
| **Refactored By** | Iphigenia Pappas |
|  |  |  |
| **6** | **Findings** | Method `System.out.println` should not be called |
| **Quality Indicator** | ***Java unwanted method calls*** |
| **Re-Engineering Rules/Standards** | Certain methods like console or debugging methods should be avoided in code because they could introduce unwanted side effects or reduce performance. |
| **Refactored By** | Heet Patel |
|  |  |  |
| **7** | **Findings** | `SimpleDateFormat` constructor should specify `Locale` |
| **Quality Indicator** | ***Avoid creating SimpleDateFormat objects without specifying the Locale*** |
| **Re-Engineering Rules/Standards** | SimpleDateFormat objects should not be initialized without specifying a Locale object. Otherwise, the default locale of the current system is used. This can lead to different results on different machines. Example for a valid object creation:  SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd.MM.yyyy", Locale.GERMANY); |
| **Refactored By** | Heet Patel |
|  |  |  |
| **8** | **Findings** | Interface comment missing |
| **Quality Indicator** | ***Interface comment completeness*** |
| **Re-Engineering Rules/Standards** | Missing interface documentation makes it hard to get a quick overview of the interface provided by a class or API. |
| **Refactored By** | Kevinkumar Patel |
|  |  |  |
| **9** | **Findings** | TODO Auto-generated method stub |
| **Quality Indicator** | ***Task tags*** |
| **Re-Engineering Rules/Standards** | Task tags (e.g. TODO) can be problematic if there is no process to manage and get rid of them. If they accumulate over time, they are better placed into an issue tracker. |
| **Refactored By** | Kevinkumar Patel |
|  |  |  |
| **10** | **Findings** | Throw of generic exception Exception |
| **Quality Indicator** | ***Imprecise Signalling*** |
| **Re-Engineering Rules/Standards** | Generic exceptions should not be thrown by methods. Appropriate subclasses should be used instead. In Java, generic exceptions comprise Throwable, Error, RuntimeException and Exception of the java.lang package. |
| **Refactored By** | Venis Patel |
|  |  |  |
| **11** | **Findings** | The `printStackTrace()` method should not be called |
| **Quality Indicator** | ***Call to printStackTrace()*** |
| **Re-Engineering Rules/Standards** | Using printStackTrace() is considered bad practice as it will only print the stack trace to standard out.In production environments this information will likely be lost. Therefore, it is recommended to use a logging framework instead. |
| **Refactored By** | Venis Patel |
|  |  |  |
| **12** | **Findings** | Clone with 2 instances of length 28 |
| **Quality Indicator** | ***Cloning*** |
| **Re-Engineering Rules/Standards** | The best strategy for removal of a clone depends on the context. Typical solutions are The extraction of utility methods, Introduction of common base classes, or application of code generators. |
| **Refactored By** | Manimaran Palani |
|  |  |  |
| **13** | **Findings** | Private field `env` is never read. |
| **Quality Indicator** | ***Avoid unused private fields (Java)*** |
| **Re-Engineering Rules/Standards** | This rule is reported when a private field in your code exists but is not used by any code path. |
| **Refactored By** | Manimaran Palani |
|  |  |  |
| **14** | **Findings** | Empty block: constructor |
| **Quality Indicator** | ***Empty blocks*** |
| **Re-Engineering Rules/Standards** | Empty blocks are typically not required and can confuse the reader as it is unclear if the empty block was intended or should be filled later on. To clarify this, developers should place an explanatory comment into the block. C++ destructors: Empty and non-virtual destructors are probably not useful. They should either be deleted or made virtual (if the class is intended for subclassing). |
| **Refactored By** | Iphigenia Pappas |
|  |  |  |
| **15** | **Findings** | Unused import: `com.userfront.domain.PrimaryTransaction` |
| **Quality Indicator** | ***Unused/duplicated imports (Java)*** |
| **Re-Engineering Rules/Standards** | Unused/Duplicated imports are not needed and make the code files unnecessarily large. Instead of manually removing them, check if your IDE can do this automatically, e.g., on each save. |
| **Refactored By** | Iphigenia Pappas |
|  |  |  |
| **16** | **Findings** | Name `com.userfront.service.UserServiceImpl` violates naming convention. Should be one of `[a-z][a-z\_0-9.]\*` |
| **Quality Indicator** | ***Java naming conventions*** |
| **Re-Engineering Rules/Standards** | Naming conventions ensure consistency in the naming and can aid code reading by supplying additional information on the type of identifier. |
| **Refactored By** | Heet Patel |
|  |  |  |
| **17** | **Findings** | Remove this use of `getBytes` |
| **Quality Indicator** | ***Classes and methods that rely on the default system encoding should not be used*** |
| **Re-Engineering Rules/Standards** | Using classes and methods that rely on the default system encoding can result in code that works fine in its "home" environment. But that code may break for customers who use different encodings in ways that are extremely difficult to diagnose and nearly, if not completely, impossible to reproduce when it's time to fix them. |
| **Refactored By** | Heet Patel |
|  |  |  |
| **18** | **Findings** | Use `BigDecimal.ZERO` instead of creating new object |
| **Quality Indicator** | ***Unnecessary creation of pre-defined `BigInteger`/`BigDecimal` object*** |
| **Re-Engineering Rules/Standards** | There are a couple pre-defined BigInteger/BigDecimal constants for certain values:  BigInteger.ZERO, BigInteger.ONE, BigInteger.TEN BigDecimal.ZERO, BigDecimal.ONE, BigDecimal.TEN Recreating these constants is thus unnecessary and should be avoided for performance reasons. |
| **Refactored By** | Kevinkumar Patel |
|  |  |  |
| **19** | **Findings** | Comment should not contain `/\*` |
| **Quality Indicator** | ***Comments should not contain nested comments*** |
| **Re-Engineering Rules/Standards** | Comments should not contain any string of other comment types, indicating nested comments. |
| **Refactored By** | Kevinkumar Patel |
|  |  |  |
| **20** | **Findings** | Catch clause catches generic exception `Exception` |
| **Quality Indicator** | ***Catch of generic exception*** |
| **Re-Engineering Rules/Standards** | Generic exceptions should not be handled by a catch block. For example, in Java, the exceptions Throwable, Error, RuntimeException, and Exception of the java.lang package count as generic. In CS, Exception, ApplicationException, or SystemException count as such. |
| **Refactored By** | Venis Patel |
|  |  |  |
| **21** | **Findings** | Logger should be specified with `UserServiceImpl.class` |
| **Quality Indicator** | ***Specify new Logger instances with current class*** |
| **Re-Engineering Rules/Standards** | The logger should be specified with the class it operates in. This way, the log output can be linked to said class. |
| **Refactored By** | Venis Patel |

# **Locations of the Source Code Undesirables**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Folder | File | Lines of Code | Source Lines of Code | Number of Findings |
| 1 | userfront/config | RequestFilter.java | 51 | 40 | 9 |
| 2 | userfront/config | SecurityConfig.java | 77 | 58 | 8 |
| 3 | userfront/controller | AccountController.java | 90 | 67 | 7 |
| 4 | userfront/controller | AppointmentController.java | 56 | 40 | 4 |
| 5 | userfront/controller | HomeController.java | 85 | 63 | 6 |
| 6 | userfront/controller | TransferController.java | 125 | 92 | 9 |
| 7 | userfront/controller | UserController.java | 50 | 34 | 3 |
| 8 | userfront/dao | AppointmentDao.java | 13 | 7 | 2 |
| 9 | userfront/dao | PrimaryAccountDao.java | 13 | 6 | 1 |
| 10 | userfront/dao | PrimaryTransactionDao.java | 13 | 7 | 2 |
| 11 | userfront/dao | RecipientDao.java | 16 | 9 | 4 |
| 12 | userfront/dao | RoleDao.java | 10 | 6 | 2 |
| 13 | userfront/dao | SavingsAccountDao.java | 13 | 6 | 1 |
| 14 | userfront/dao | SavingsTransactionDao.java | 14 | 7 | 2 |
| 15 | userfront/dao | UserDao.java | 14 | 9 | 4 |
| 16 | userfront/domain/security | Authority.java | 19 | 12 | 2 |
| 17 | userfront/domain/security | Role.java | 50 | 32 | 4 |
| 18 | userfront/domain/security | UserRole.java | 59 | 39 | 5 |
| 19 | userfront/domain | Appointment.java | 86 | 67 | 1 |
| 20 | userfront/domain | PrimaryAccount.java | 66 | 46 | 2 |
| 21 | userfront/domain | PrimaryTransaction.java | 108 | 82 | 5 |
| 22 | userfront/domain | Recipient.java | 86 | 65 | 1 |
| 23 | userfront/domain | SavingsAccount.java | 63 | 46 | 2 |
| 24 | userfront/domain | SavingsTransaction.java | 106 | 82 | 5 |
| 25 | userfront/domain | User.java | 208 | 158 | 4 |
| 26 | userfront/resource | AppointmentResource.java | 34 | 25 | 3 |
| 27 | userfront/resource | UserResource.java | 55 | 43 | 4 |
| 28 | userfront/service/UserServiceImpl | AccountServiceImpl.java | 112 | 81 | 10 |
| 29 | userfront/service/UserServiceImpl | AppointmentServiceImpl.java | 36 | 26 | 6 |
| 30 | userfront/service/UserServiceImpl | TransactionServiceImpl.java | 145 | 110 | 15 |
| 31 | userfront/service/UserServiceImpl | UserSecurityService.java | 33 | 25 | 2 |
| 32 | userfront/service/UserServiceImpl | UserServiceImpl.java | 121 | 91 | 16 |
| 33 | userfront/service | AccountService.java | 18 | 12 | 7 |
| 34 | userfront/service | AppointmentService.java | 16 | 9 | 5 |
| 35 | userfront/service | TransactionService.java | 36 | 22 | 13 |
| 36 | userfront/service | UserService.java | 32 | 18 | 12 |
| 37 | userfront | UserFrontApplication.java | 13 | 9 | 2 |
|  |  | ***Total Lines of Code*** | **2142** | ***Total Findings*** | **190** |

# **Software Metric Log**

# **Refactoring Report**

# **Software Specifications**

# **Refactored Source Code of R**

# **References**