Report on Refactoring project Chemistry Calculator

Submitted to
Dipok Chandra Das
Assistant Professor, IIT, NSTU

Submitted by

Md. Mynuddin (ASH1825007M)
Al Adnan Sami (ASH1825008M)
Shuvra Adiyta (ASH1825009M)
Al Jaber Nishad (ASH1825029M)
Md. Kamruzzaman Shekh (ASH1825035M)

Introduction

Refactoring refers to the systematic process of modifying an existing codebase without altering its external behavior, with the aim of enhancing its readability, maintainability, and extensibility. The main objective of refactoring is to simplify the code and make it more comprehensible to developers, so that they can easily modify and improve it.

In this report, we will discuss the refactoring of a project named **Chemistry Calculator**, which provides many functionalities like-

- Equation balancing
- Electron configuration
- Molar mass
- Calculations of concentration and titration

In this document the refactoring process are performed by following these steps-

- Identification
- Selection
- Application
- Evaluation

Identification

The Chemistry Calculator project was deemed a suitable candidate for refactoring owing to various factors, including the large size of the codebase, the intricate nature of the calculations, and the absence of a coherent structure in the code. The code was laborious to maintain, characterized by redundant and superfluous statements that impeded its comprehension and modification. Additionally, the code manifested several problems, such as inadequate naming conventions, lack of modularity, and repetitive code. Moreover, the project lacked adequate documentation and testing, which rendered it difficult to confirm the precision of the calculations.

Selection

The refactoring process was initiated by conducting an extensive examination of the existing codebase, which resulted in the identification of several areas for improvement. The primary emphasis was on eliminating redundant code, streamlining complex calculations, and enhancing the overall structure and organization of the code. A gradual refactoring approach was employed, whereby modifications

were made incrementally to ensure that the code continued to function properly and produce accurate results throughout the process.

Below are the code smells that are found in the project codebase-

- Remove Unused Imports
- Primitive Obsession
- Long Method
- Long Parameter List
- Large Class
- Comments
- Duplicated Code
- Inappropriate Naming
- Divergent Change
- Speculative Generality
- Dead Code
- Shotgun Surgery
- Inappropriate Intimacy

Class Name	Code Smell
Atom	Primitive Obsession, Large Class, Speculative Generality
Compound	Inappropriate Naming, Comments
CompoundManager	Speculative Generality
Concentration	
Converter	
Database	Speculative Generality,
DatabaseSerializer	Speculative Generality
EquationBalancer	Inappropriate Naming , Long Method

Fraction	Speculative Generality
History	Inappropriate Naming, Speculative Generality
InsufficientDataException	
InvalidAtomException	
InvalidEquationException	
Matrix	Inappropriate Naming, Long Method
Titration	Inappropriate Naming
Orbitals	Speculative Generality
ConcentrationPanel	Primitive Obsession, Dead Code, Speculative Generality, Large Class
ElectronConfigPanel	Primitive Obsession, Dead Code, Speculative Generality , Large Class
EquationBalancePanel	Primitive Obsession, Dead Code, Speculative Generality, Large Class
Formater	
FxPieChart	
HistoryFrame	Primitive Obsession, Dead Code, Speculative Generality
Home	Inappropriate Naming, Long Method
MolarMassPanel	Primitive Obsession, Dead Code, Speculative Generality
NeedHelpPanel	Primitive Obsession, Dead Code
PercentOfCompletionPanel	Speculative Generality,
Sidebar	Primitive Obsession, Dead Code, Large Class
TitrationPanel	Primitive Obsession, Dead Code, Speculative Generality

Table 1: Code smell found for each class

Application

The refactoring process involved several steps, starting with

- Revamping each file by incorporating appropriate indentation and eliminating superfluous blank lines to enhance its readability
- Eliminating redundant code, including previously commented-out code sections that were used for debugging purposes
- Addressing each code smell that was identified during the selection process by implementing suitable remedies

Class Name	Remedy Applied
Atom	Extract method, Extract Class, Remove Code
Compound	Extract Method, Remove Code
CompoundManager	Extract Method
Database → AtomDatabase	Extract Class, Extract Method, Remove Code, Rename, Remove Code
EquationBalancer	Extract Method
History → HistoryManager	Inline method
Matrix	Extract Method
Titration	Extract Method
ConcentrationPanel	Extract Method, Replace Data value with Object, Remove Code, Remove Parameters
ElectronConfigPanel	Extract Method, Replace Data value with Object, Remove Code, Remove Parameters
EquationBalancePanel	Extract Method, Replace Data value with Object, Remove Code, Remove Parameters
Fraction	Remove Code
HistoryFrame	Extract Method, Replace Data value with Object, Remove Code
Home	Extract Method, Move Method, Move Field
MolarMassPanel	Replace Data value with Object, Extract Method, Remove Code

NeedHelpPanel	Extract Method, Remove Code
PercentOfCompletionPanel	Replace array with object, Extract Method, Extract Class, Remove Code
Sidebar	Extract Method, Move Method
TitrationPanel	Extract Method, Extract Class, Remove Code

Table 2: Code smell remedy for each class

Evaluation

To evaluate whether refactored code improves maintainability, readability and modularity some of the techniques are-

1. Code Quality Metrics
2. Peer Review
3. Unit Testing
4. User Feedback

For **Chemistry Calculator** project evaluation we performed code quality metrics.

Code Quality Metrics

Code quality metrics are instrumental in assessing the quality of software code. During the refactoring process of the Chemistry Calculator project, several commonly used code quality metrics were employed, including:

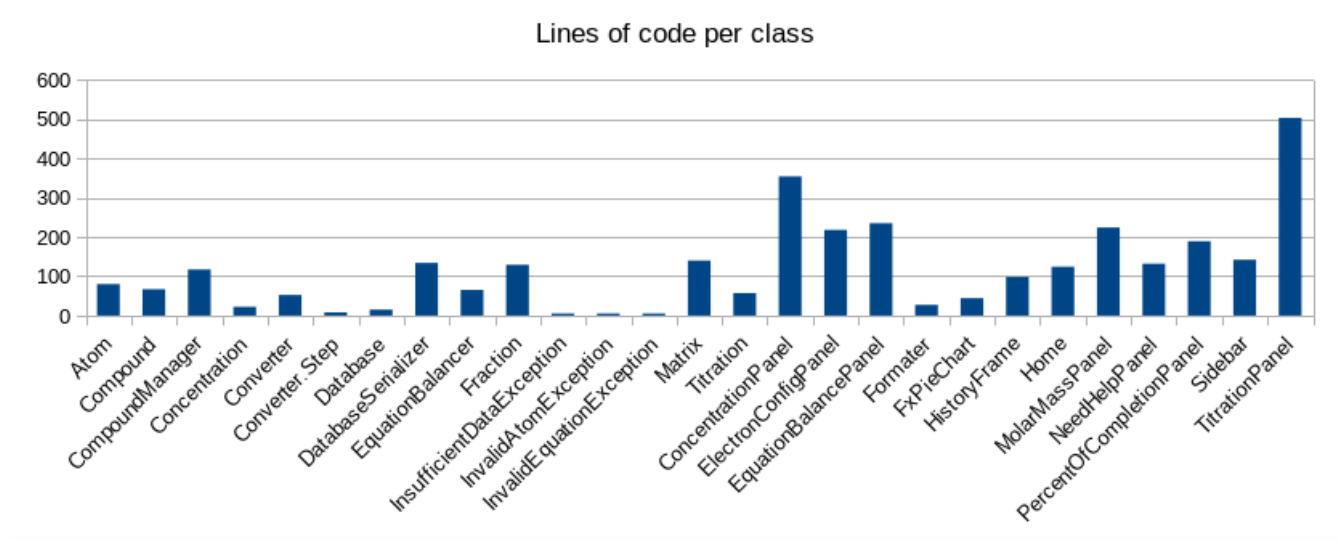
• **Weighted Methods per Class (WMC):** This metric determines the complexity of a class in object-oriented programming by computing the number of methods in the class and assigning a weight to each method based on its complexity.

• **Cyclomatic complexity:** This metric gauges the complexity of a program by counting the number of decision points in the code.

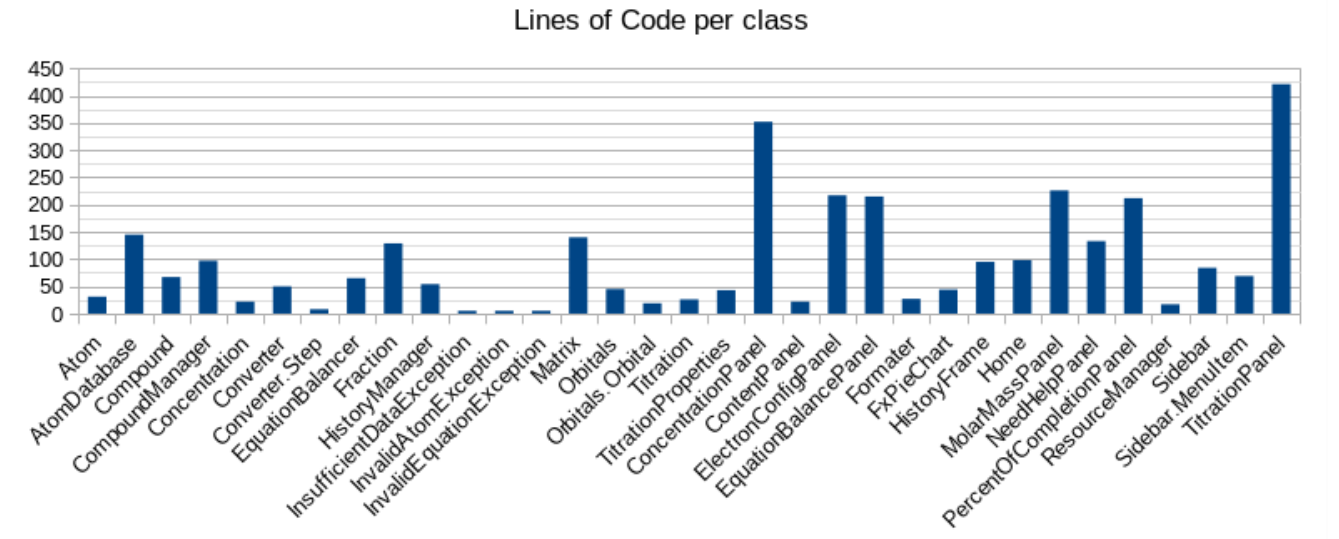
• **Lines of code (LOC):** This metric quantifies the number of lines of code present in a program.

These quality metrics performed with **MetricesReloaded** plugin from **Intellij Idea (Community Edition)** software.

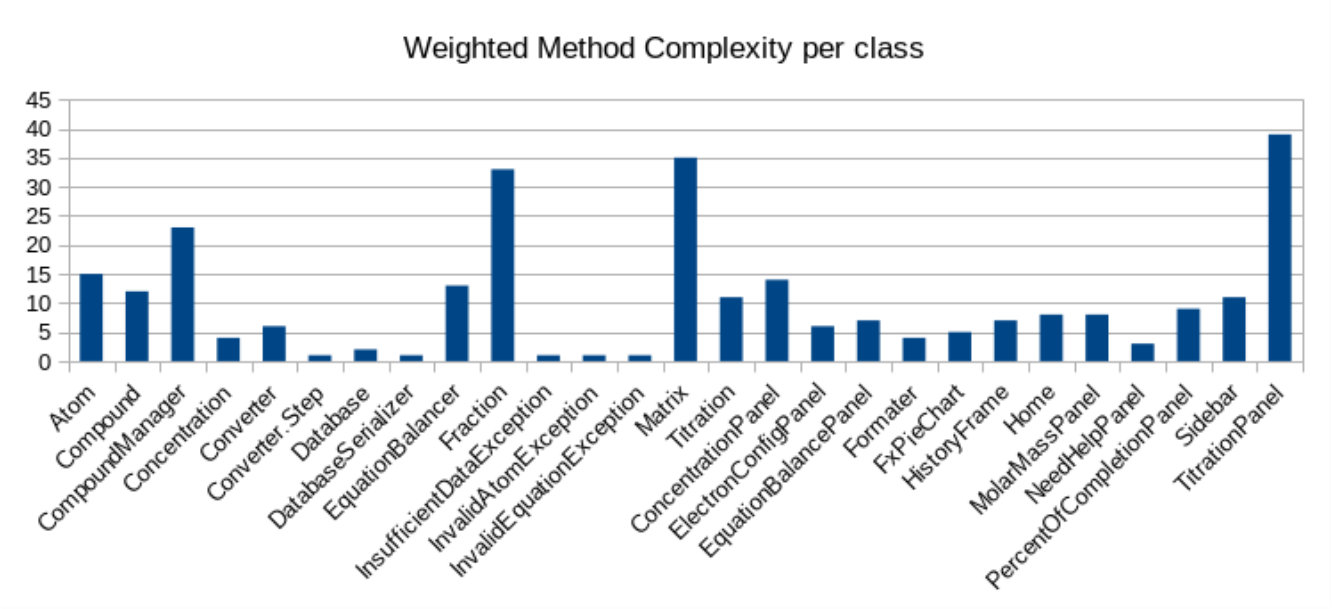
Before refactoring the lines of code per class is shown below



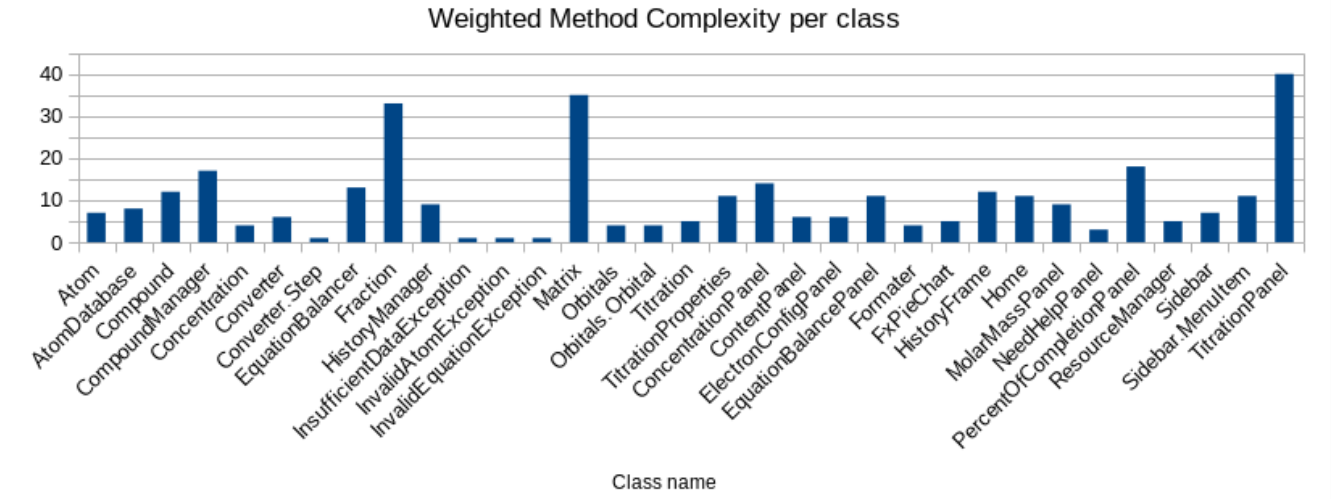
After refactoring the lines of code per class is



Before refactoring the weighted method per class is



After refactoring the weighted method per class is



Conclusion

To summarize, the refactoring process of the Chemistry Calculator project was a success, as it significantly enhanced the readability, maintainability, and extensibility of the codebase. The gradual refactoring approach that was adopted ensured that the code remained functional and precise throughout the process. The refactored code was evaluated and observed to be more readable, maintainable, and extensible than the original code. As a result, the refactoring process produced a more dependable and maintainable project that will be easier to modify and expand in the future.