

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

C++ Code Refactoring Methods

Доповідь виконав:

студент III курсу

групи ПЗПІ-22-2

Береговий Даніїл Олександрович

Перевірив

Доц. кафедри ПІ

Лещинський Володимир Олександрович

2024

Мета роботи:

Навчитися застосовувати методи рефакторингу коду на практиці, щоб робити свої програми зрозумілішими та зручнішими для підтримки. Працюючи з власними прикладами, я зможу краще зрозуміти, як покращувати структуру коду, не змінюючи його функціональність.

Хід роботи:

Спершу я ознайомився з методами рефакторингу, описаними у книзі Мартіна Фаулера «Refactoring. Improving the Design of Existing Code», і обрав три, які найкраще підходять для покращення якості коду: Extract Method, Rename Method і Simplify Conditional Expression. Ці методи здалися мені найбільш доречними для вирішення проблем, які я виявив у своїх проєктах, таких як дублювання коду, некоректні назви функцій та складна логіка умов.

Проаналізувавши свій код із лабораторних робіт, я виділив кілька фрагментів, які потребували вдосконалення. Наприклад, у деяких місцях повторювалися однакові блоки коду, що ускладнювало підтримку програми. В інших випадках функції мали назви, які не пояснювали їхнього призначення, а це могло заплутати навіть мене самого через певний час. Також були складні умовні вирази, які було важко зрозуміти з першого погляду.

Я застосував обрані методи рефакторингу до конкретних ділянок коду. Спочатку за допомогою Extract Method виділив повторюваний блок коду в окрему функцію, що зробило основну функцію коротшою та зрозумілішою. Потім використав Rename Method, щоб перейменувати функцію з неінформативною назвою, що одразу покращило зрозумілість коду. Нарешті, за допомогою Simplify Conditional Expression спростив складний вкладений умовний вираз, замінивши його на простішу та логічнішу конструкцію.

Для демонстрації результатів я підготував слайди, у яких детально описав кожен метод рефакторингу. У слайдах були представлені фрагменти коду до і після змін, пояснення, чому саме цей метод було застосовано, а також переваги, які він забезпечив. На практичному занятті я презентував свої результати та обговорив їх із викладачем та одногрупниками, враховуючи їхні коментарі та поради.

Висновки:

Рефакторинг дозволив покращити структуру мого коду, зробивши його більш читабельним та легким у підтримці. Методи Extract Method, Rename Method, та Simplify Conditional Expression допомогли усунути дублювання, покращити назви функцій, і спростити складну логіку, що підвищило якість та ефективність роботи з кодом.

Рефакторинг коду на C++






Виконав: Береговий Даніїл
Олександрович
Група: ПЗПІ-22-2
Перевірив: Лещинський Володимир
Олександрович

Покращення якості коду за допомогою методів рефакторингу

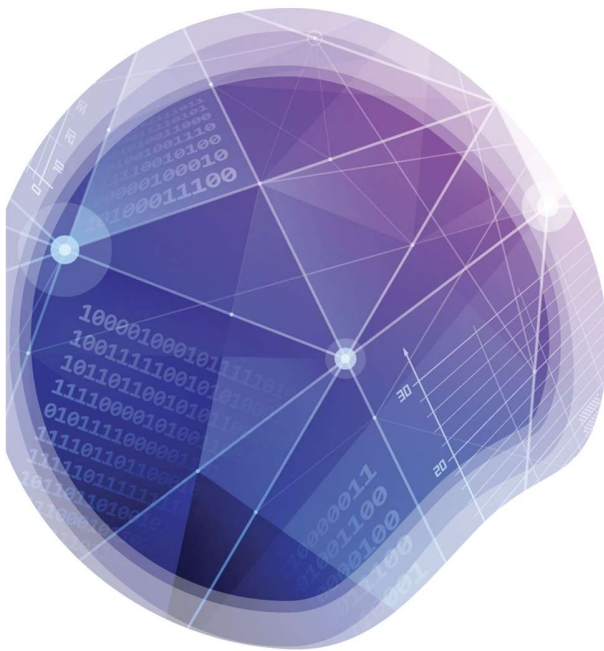
1



ЗМІСТ

-  1. Вступ до рефакторингу
-  2. Replace Magic Number with Symbolic Constant
-  3. Move Field
-  4. Remove Assignments to Parameters
-  5. Висновки та переваги рефакторингу

2

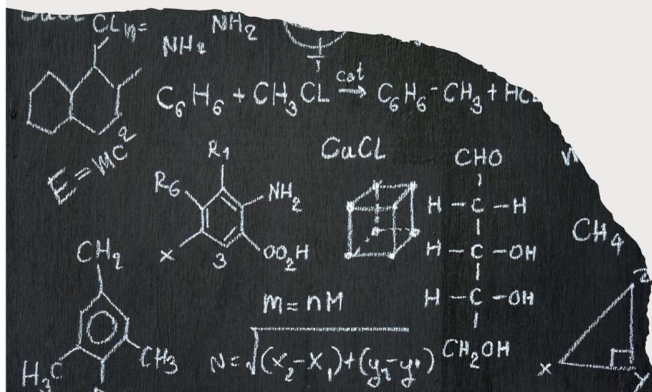


Вступ до рефакторингу

- **Визначення:** Рефакторинг - це процес реструктуризації існуючого коду без зміни його зовнішньої поведінки
- **Мета:** Покращити читабельність коду, зручність супроводу та масштабованість.
- **Головне:** Реальні приклади трьох технік рефакторингу в C++.

3

Replace Magic Number with Symbolic Constant



Проблема: Захардкоджені «магічні числа» зменшують читабельність коду та призводять до помилок під час оновлень.

Приклад (Перед Рефакторингом):

```
double calculateCircleArea(double radius) {
    return radius * radius * 3.14159; // Magic number }
```

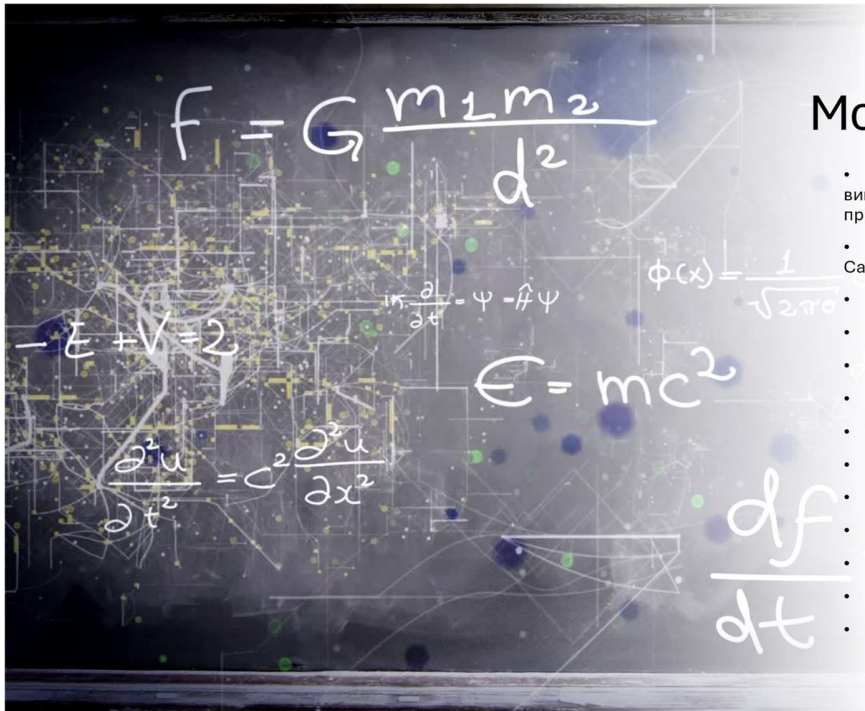
Вирішення: Замінити магічні числа іменованими константами.

Приклад (Після Рефакторингу):

```
const double PI = 3.14159;
double calculateCircleArea(double radius) {
    return radius * radius * PI; }
```

Переваги: Покращена читабельність та централізоване керування константами.

4



Move Field

- Проблема: Поле (змінна-член) використовується переважно іншим класом, що призводить до поганої інкапсуляції.
- Вирішення: Перемістіть поле 'oilLevel' до класу Car.
- Приклад (Перед Рефакторингом):


```

class Engine {
    double oilLevel;
}
class Car {
    Engine engine;
    void checkOil() {
        if (engine.oilLevel < 2.0) {
            // Action }}}
      
```
- Приклад (Після Рефакторингу):


```

class Car {
    double oilLevel;
      
```

5



Проблема: зміна параметрів функції може призвести до несподіваних побічних ефектів.

Рішення: замість цього використовуйте локальну змінну.

Приклад (Перед Рефакторингом):

- void updateValue(int param) {
- param += 10;
- }

Приклад (Після Рефакторингу):

- void updateValue(int param) {
- int result = param + 10;
- }

6

Короткий опис переваг

1. Replace Magic Number with Symbolic Constant : покращення читабельності.
2. Move Field: Краща інкапсуляція.
3. Remove Assignments to Parameters: Допомогає уникати побічних ефектів.

7

Висновки

- Вплив рефакторингу :
- - Покращена читабельність коду та зручність обслуговування.
- - Простіші майбутні модифікації та зменшення технічного боргу.
- Наступні кроки: застосуйте ці методи до своїх проектів.

8