МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Бахмутський навчально-науковий професійно-педагогічний інститут

Кафедра електромеханічних та комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ Інна НЕФЬОДОВА

(підпис) (ім'я, прізвище)

«02» грудня 2024 року

КВАЛІФІКАЦІЙНА РОБОТА (ПРОЄКТ)

рівень вищої освіти другий (магістерський)

спеціальність 015.39 Професійна освіта (Цифрові технології)

освітньо-професійна програма Професійна освіта. Комп'ютерні технології в управлінні та навчанні

тема «Професійна підготовка фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти»

Виконав(ла)

здобувач(ка) групи БД-К23мг
(шифр групи)

Іван ПАНЧУК _____
(ім'я, прізвище) (підпис)

Керівник роботи

к.т.н., доц. Павло ЧИКУНОВ _____
(науковий ступінь, вчене звання, ім'я, прізвище) (підпис)

Рецензент роботи

к.пед.н., доц. Наталія ЛОГІНОВА _____
(науковий ступінь, вчене звання, ім'я, прізвище) (підпис)

Консультант

к.пед.н., доц. Юлія БОБРИКОВА _____
(науковий ступінь, вчене звання, ім'я, прізвище) (підпис)

Засвідчую, що у цій роботі немає цитат та вилучень з праць інших авторів без відповідних посилань

здобувач (ка) _____
(підпис)

Харків – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет/ННІ Бахмутський навчально-науковий професійно-педагогічний інститут
Кафедра Електромеханічних та комп'ютерних систем
Рівень вищої освіти другий (магістерський)
Спеціальність 015.39 Професійна освіта (Цифрові технології)
Освітньо-професійна програма Професійна освіта. Комп'ютерні технології в управлінні та навчанні

ЗАТВЕРДЖУЮ

Завідувач кафедри
(підпис) Інна НЕФЬОДОВА (ім'я, прізвище)
«08» жовтня 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Панчук Іван Миколайович
(прізвище, ім'я, по батькові здобувача)

1. Тема роботи Професійна підготовка фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти
керівник роботи Чикунов Павло Олександрович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «08» жовтня 2024 року № 5101-5/3236

2. Строк подання здобувачем роботи «02» грудня 2024 р.

3. Перелік питань, які потрібно розробити: Актуальність професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти. Характеристика об'єктів галузі: стан і стратегії розвитку. Вимоги до кадрового забезпечення об'єкту галузі. Методика професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти.

4. План роботи

№ з/п	Назви етапів роботи
1	Огляд літературних джерел, нових розробок, опублікованих даних та іншої інформації, пов'язаної з темою роботи.
2	Дослідження теоретичних підходів до актуальності професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти».
3	Характеристика об'єктів галузі: стан і стратегії розвитку.
4	Розробка методики професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти.
5	Розробка вимог до кадрового забезпечення об'єкту галузі
6	Оформлення першого варіанту тексту, подання його на ознайомлення науковому керівнику
7	Усунення недоліків, написання остаточного варіанту тексту, оформлення дипломної роботи
8	Подання роботи на кафедру, перевірка на плагіат та зовнішнє рецензування роботи
9	Захист дипломної роботи у ЕК

5. Дата видачі завдання «08» жовтня 2024 р.

Здобувач(ка) _____ Іван ПАНЧУК
(підпис) (ім'я, прізвище)

Керівник роботи _____ Павло ЧИКУНОВ
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Мета дослідження – теоретично обґрунтувати та частково перевірити методику професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду».

Об'єктом дослідження роботи є процес професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

Предметом дослідження роботи є методика професійної підготовки фахівців з цифрових технологій до розробки комплексу цифрових освітніх для викладання освітнього модуля «Рефакторинг програмного коду».

Охарактеризовано систему професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти.

Виконано аналіз характеристик об'єктів галузі проєктування програмного забезпечення. Виконана постановка 3 нових лабораторних робіт. Виконано аналіз вимог до кадрового забезпечення об'єкту ІТ-галузі в умовах цифрової трансформації суспільства.

Розроблено дидактичний проект консультативного заняття з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти.

За основними результатами дослідження виконана публікація тези доповіді на VIII міжнародній НПК «Студенти та молодь – для майбутнього країни» (Бахмут-Харків, 14-15 листопада 2024 р.).

Обсяг дипломної роботи становить: пояснювальна записка, презентація доповіді. Пояснювальна записка складається з вступу, чотирьох розділів, висновків, списку використаних джерел, додатків. Загальний обсяг роботи 83 сторінок, з яких 59 сторінок основного тексту. Список використаних джерел становить 61 найменувань, 9 таблиць, 29 рисунків.

ПРОФЕСІЙНА ПІДГОТОВКА, ЛАБОРАТОРНИЙ ПРАКТИКУМ, РЕФАКТОРИНГ ПРОГРАМНОГО КОДУ, КАДРОВЕ ЗАБЕЗПЕЧЕННЯ, МЕТОДИЧНА РОЗРОБКА

ABSTRACT

The aim of the research is to theoretically substantiate and partially test the methodology for professional training of specialists in digital technologies to teach the educational module "Code Refactoring."

The object of the research is the process of professional training of specialists in digital technologies for teaching the educational module "Code Refactoring" in higher education institutions.

The subject of the research is the methodology for professional training of specialists in digital technologies to develop a set of digital educational tools for teaching the educational module "Code Refactoring."

The study characterizes the system of professional training of specialists in digital technologies for teaching the educational module "Code Refactoring" in higher education institutions.

An analysis of the characteristics of objects in the field of software design was conducted. Three new laboratory works were proposed. An analysis of requirements for personnel support in the IT sector under conditions of digital transformation of society was carried out.

A didactic project for a consultative session on the topic "Performing Refactoring Techniques in Visual Studio," as part of the course "Code Analysis and Refactoring," was developed for higher education students.

The main research findings were presented as a thesis report at the VIII International Scientific and Practical Conference "Students and Youth – for the Future of the Country" (Bakhmut-Kharkiv, November 14–15, 2024).

The scope of the thesis includes: an explanatory note and a presentation for the report. The explanatory note consists of an introduction, four chapters, conclusions, a list of references, and appendices. The total volume of the work is 83 pages, of which 59 pages are the main text. The list of references contains 61 entries, 9 tables, and 29 figures.

PROFESSIONAL TRAINING, LABORATORY PRACTICE, CODE REFACTORING, PERSONNEL SUPPORT, METHODOLOGICAL DEVELOPMENT

ЗМІСТ

ВСТУП

Світ стрімко переходить у цифрову епоху, і його мешканці мають опановувати цифрові технології та ефективно застосовувати їх у різних сферах, таких як освіта, наука та бізнес. У таких умовах ключовим фактором стає доступ до знань, що зберігаються у спеціалізованих середовищах і можуть бути доступними з будь-якого місця й у будь-який час. Сучасний фахівець із цифрових технологій повинен постійно пристосовуватися до швидких змін навколишнього середовища, спричинених еволюцією комп'ютерних технологій. Успішність такої адаптації визначає його професійний успіх, кар'єрний ріст і можливості для самореалізації.

Особливо важливим є професійна підготовка фахівців цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти. Вона спрямована на формування компетентностей майбутніх фахівців відповідно до сучасних вимог освітнього процесу. Водночас, варто зазначити, що, по-перше, досвід такої підготовки залишається недостатньо вивченим. Тому пріоритетним завданням є створення умов для ефективної професійної підготовки, зокрема через розвиток системи післядипломної неперервної освіти, яка сприяє підтримці населення під час зміни професійної діяльності. Аналіз теорії та практики свідчить про те, що реалізація програм професійної підготовки фахівців все ще потребує додаткового дослідження. Як виявлено на основі аналізу наукової літератури, ученими досліджено окремі аспекти порушеної проблеми. Науково-методологічним аспектам освітньої сфери присвячені роботи вчених О. Антонюка, В. Бакуменка, О. Батанова , В. Гриценко, Ю. Журавльової, А. Кобця, В. Коврегіна, В. Кременя, В. Лугового, В. Мороза , В. Огаренка та ін.; різні аспекти професійної підготовки майбутніх інженерів-педагогів досліджують Е. Абільтарова, І. Васильєва, Н. Волкова, Н. Брюханова, Р. Горбатюк, О. Коваленко, М. Лазарєв, В. Кулешова, Л. Штефан та ін.

Актуальність дослідження й вирішення поставленої проблеми, а також

вивчення педагогічного досвіду обумовлені існуючими суперечностями, зокрема:

– між сучасними вимогами до організації самоосвітньої діяльності майбутніх фахівців у галузі цифрових технологій та недостатнім використанням можливостей для формування їхньої самоосвітньої компетентності в освітньому процесі;

– між потребою у формуванні самоосвітньої компетентності фахівців і недостатньою розробленістю відповідних методичних підходів.

Подолання цих суперечностей стало основою для визначення теми дослідження: «Професійна підготовка фахівців із цифрових технологій для викладання освітнього модуля "Рефакторинг програмного коду" у закладах вищої освіти».

Мета дослідження – теоретично обґрунтувати та частково перевірити методику професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

У зв'язку з поставленою метою, в роботі вирішуються наступні завдання:

1. Проаналізувати та визначити ступінь актуальності проблеми професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

2. Обґрунтувати теоретичну і методичну базу концепції методичної системи професійної підготовки майбутніх фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

Об'єкт дослідження: процес професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

Предмет дослідження: методика професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

В ході написання роботи використані такі методи дослідження:

– теоретичні методи (аналіз, синтез, порівняння, моделювання, узагальнення) потрібні для вивчення психолого-педагогічної літератури і визначення концептуальних засад дослідження, уточнення сутності й особливостей процесу професійної підготовки майбутніх фахівців з цифрових технологій;

– емпіричні методи (анкетування, бесіди з учасниками експерименту, педагогічне спостереження, самооцінювання, тестування) потрібні для визначення рівнів сформованості компетентності майбутніх фахівців з цифрових технологій.

Наукова новизна одержаних результатів дослідження полягає в теоретичному обґрунтуванні структури професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти та в уточненні сутності поняття «професійна підготовка» фахівців з цифрових технологій.

Подальшого розвитку набули зміст навчання майбутніх фахівців з цифрових технологій та методика організації роботи для формування самоосвітньої компетентності.

Теоретичне та практичне значення одержаних результатів полягає в розробці методики професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти, обґрунтуванні змісту, форм та методів організації процесу підготовки фахівців з цифрових технологій.

Матеріали дослідження використовувались при підготовці здобувачів вищої освіти Навчально-наукового професійно-педагогічного інституту УІПА (м. Бахмут) зі спеціальності 015 Професійна освіта (Цифрові технології).

Апробація результатів дослідження: за основними результатами дослідження виконана доповідь на міжнародній науково-практичній конференції здобувачів вищої освіти та молодих учених «Студенти та молодь – для майбутнього країни» (Бахмут-Харків, 17 листопада 2024 р.).

РОЗДІЛ 1 АКТУАЛЬНІСТЬ ПРОФЕСІЙНОЇ ПІДГОТОВКИ ФАХІВЦІВ З ЦИФРОВИХ ТЕХНОЛОГІЙ ДЛЯ ВИКЛАДАННЯ ОСВІТНЬОГО МОДУЛЯ «РЕФАКТОРИНГ ПРОГРАМНОГО КОДУ» У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ

На сучасному етапі розвитку України питання створення ефективних, науково обґрунтованих механізмів реалізації державної кадрової політики набуло особливої важливості. Це не лише основа формування національної ідентичності, але й ключ до побудови сталого та конкурентоспроможного державного управління. Професійно підготовлені кадри є головним ресурсом держави, носієм її менталітету, культури та історичного досвіду.

Якісна професійна підготовка фахівців є однією з головних умов соціально-економічного розвитку країни. Цей процес має базуватися на інтеграції сучасних інформаційно-комунікаційних технологій (ІКТ) у навчальні програми закладів освіти. Університети та інші освітні установи повинні створювати цілісне навчальне середовище, яке дозволяє готувати студентів до викликів сучасного ринку праці.

Зважаючи на стрімкі зміни в ІТ-індустрії, для підготовки фахівців з цифрових технологій недостатньо лише розвитку професійних, комунікативних чи дослідницьких навичок. У таких спеціалістів необхідно формувати самоосвітню компетентність, яка забезпечує їх здатність до безперервного навчання та вдосконалення. Це особливо важливо для забезпечення їхньої конкурентоспроможності на ринку праці в умовах інтенсивного розвитку цифрових технологій.

Особливість підготовки фахівців з цифрових технологій, які викладатимуть модуль «Рефакторинг програмного коду» у закладах вищої освіти, полягає в необхідності постійного вдосконалення їхніх знань. Такі спеціалісти мають бути готові до освоєння нових технологій та використання їх у своїй професійній діяльності. Це вимагає не лише ґрунтовних знань з інформаційних технологій, але й розвинених навичок самонавчання та

адаптації до змін.

Така підготовка має базуватися на комплексному підході, який включає як традиційні методи навчання, так і використання сучасних цифрових платформ. Крім того, слід активно розвивати співпрацю між закладами освіти та роботодавцями, щоб адаптувати навчальні програми до актуальних вимог ринку праці [49].

Сучасний етап розвитку науки та техніки формує технологічне та інформаційне середовище, в якому функціонує людина, що впливає на її взаємодію з природним та соціальним оточенням, а також визначає межі її можливостей на поточному етапі науково-технічного прогресу. Аналіз соціальних процесів дозволяє прогнозувати майбутній розвиток освітньої системи, структура, склад і характер діяльності якої повинні відповідати вимогам науки, техніки та соціального середовища, а також внутрішнім цілям самої освіти. Вона передбачає процес і результат засвоєння знань, практичних умінь і навичок, що визначають розумовий, пізнавальний та творчий потенціал особистості, а також її морально-естетичну культуру, що разом формує соціальне обличчя та індивідуальну своєрідність людини.

У цьому контексті важливим стає питання переосмислення науково-теоретичних основ та практичних аспектів підготовки фахівців з цифрових технологій для викладання модуля «Рефакторинг програмного коду» в умовах вищої освіти. Це повинно ґрунтуватися на компетентнісному підході та впровадженні нових методичних підходів. Потреба у таких фахівцях, що володіють розвиненою оцінювальною компетентністю, зростає, оскільки це є важливим фактором особистісного розвитку, професійного зростання та успішної життєдіяльності [37].

Підготовка кваліфікованих фахівців у сучасних соціально-економічних умовах ставить перед системою професійної освіти фахівців з цифрових технологій для викладання модуля «Рефакторинг програмного коду» у закладах вищої освіти низку вимог. Процес формування професійної компетентності таких фахівців має надзвичайно важливе значення, оскільки є

складовою професійної культури. Це визначає соціальну орієнтацію особистості у її професійній діяльності, яка включає не тільки сукупність знань і навичок, але й сформовану професійну та особисту позицію, здатність до самооцінки. Це сприяє активному та творчому підходу до роботи, розвитку важливих особистісних і професійних якостей, які визначають ефективність діяльності фахівця і його самореалізацію [50].

Ґрунтовною основою вивчення та розв'язання проблеми дослідження теоретичних та методичних засад підготовки фахівців з цифрових технологій стали результати напрацювань відомих науковців з різних напрямів освіти: проблем філософії освіти (В. Андрущенко, І. Зязюн, В. Кремень, В. Лутай); системного підходу до організації освітнього процесу (В. Кузьмін, Є. Юдін та ін.); психології освіти (Г. Балл, Л. Виготський, О. Леонтьєв, Ю. Самарін, В. Семиченко, Н. Тализіна та ін.); педагогіки професійної освіти (В. Безрукова, Р. Гуревич, О. Дубасенюк, О. Дубинчук, Н. Кузьміна, Л. Лук'янова, В. Мадзігон, Н. Ничкало, Л. Оршанський, Л. Сидорчук, В. Тименко, А. Цина); структурування знань у змісті освіти (Б. Гершунський, В. Гінецинський, В. Лєдньов, О. Щербак, та ін.); інтеграції технологій в освітній процес (О. Білик, М. Корець, Д. Корчевський, М. Піддячий та ін.); застосування цифрових технологій в освіті (О. Авраменко, В. Биков, Т. Бодненко, Т. Вакалюк, І. Войтович, А. Гедзик, Ю. Горошко, А. Гуржій, М. Жалдак, Л. Карташова, В. Лапінський, Л. Макаренко, Н. Морзе, Ю. Рамський, С. Семеріков, О. Спірін, Г. Ткачук, Ю. Триус, В. Франчук, С. Яшанов та ін.); різні аспекти професійної підготовки майбутніх інженерів-педагогів (Е. Абільтарова, І. Васильєва, Н. Волкова, Н. Брюханова, Р. Горбатюк, О. Коваленко, В. Кулешова, В. Мальована, М. Лазарєв, С. Хоменко, Л. Штефан та ін.); процес формування комунікативної компетентності та її складових у здобувачів освіти закладів інженерної та інженерно-педагогічної освіти (Т. Калініченко, К. Ковальова, В. Кручек та ін.).

Аналіз наукової літератури з цієї теми показує, що деякі аспекти професійної підготовки фахівців з цифрових технологій для викладання

модуля «Рефакторинг програмного коду» у вищих навчальних закладах залишаються недостатньо дослідженими. Зокрема, не приділяється достатньо уваги теоретичним та практичним аспектам впровадження компетентнісного підходу у вищу освіту та підготовку спеціалістів у цій сфері. Це вказує на необхідність поглиблених досліджень та розробки нових методик для ефективного формування відповідних професійних компетенцій [45].

Вища освіта повинна сприяти підготовці кваліфікованих та компетентних фахівців для різних секторів економіки. У сучасних умовах розвитку освітніх систем важливо не лише зберігати сильні сторони професійно-технічної освіти, але й робити її більш гнучкою та адаптивною, щоб вона відповідала вимогам суспільства та ринку праці [46].

У зв'язку з цим постає питання перегляду основних науково-теоретичних підходів і практичних аспектів професійної підготовки фахівців з цифрових технологій для викладання модуля «Рефакторинг програмного коду» у вищих навчальних закладах, що базуються на компетентнісному підході та інтеграції нових методичних концепцій.

Зі зростанням важливості цих процесів виникає потреба у фахівцях, які володіють високим рівнем оцінювальної компетентності, оскільки це є важливим фактором для їх особистісного розвитку, професійного зростання та успішної життєдіяльності в цілому.

Освітній процес у підготовці фахівців з цифрових технологій для викладання модуля «Рефакторинг програмного коду» має сприяти розвитку пізнавальної активності студентів, стимулюванню їх інтелектуального потенціалу, культурних і моральних цінностей, а також формуванню цілісної особистості, готової до сучасних професійних викликів. Метою освіти є не лише передача знань, умінь і навичок, а й розвиток кругозору, здатності до міждисциплінарних рішень, самонавчання, а також формування гуманістичних цінностей [50].

Для підвищення ефективності освітнього процесу в системі професійної підготовки і відповідності сучасним вимогам модернізації освіти, потрібно

враховувати сучасні інтерпретації принципу неперервності з концептуальним осмисленням специфічної природи нового виду професійної діяльності.

Отже, професійна підготовка фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти потребує вирішення проблеми, яку ми вбачаємо через теоретичне обґрунтування методики професійної підготовки.

Висновки до розділу

У кваліфікаційній роботі відповідно до мети і завдань розкрито стан наукової проблеми.

У ході дослідження уточнено та систематизовано основні поняття, що характеризують сутність та структуру процесу професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти та виділено пріоритетні напрямки удосконалення професійних компетентностей.

З’ясовано, що професійна підготовка фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти є актуальною у професійній педагогіці.

Проаналізовано ступінь актуальності проблеми професійної підготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

Охарактеризовано систему професійної подготовки фахівців з цифрових технологій для викладання освітнього модуля «Рефакторинг програмного коду» у закладах вищої освіти.

РОЗДІЛ 2 ХАРАКТЕРИСТИКА ОБ'ЄКТІВ ГАЛУЗІ: СТАН І СТРАТЕГІЇ РОЗВИТКУ

2.1 Аналіз силабусів навчальних дисциплін

Силабус навчальної дисципліни «Аналіз та рефакторинг коду» освітньо-професійної програми «Інженерія програмного забезпечення» Національного університету «Одеська юридична академія» [1] своєю метою визначає ознайомлення здобувачів освіти з методологією аналізу та рефакторінгу коду об’єктно-орієнтованих програм, зокрема з методами інспекції та аналізу «брудного» коду, прийомами переробки коду без зміни функціональності, шаблонами типових помилок.

Завдання навчальної дисципліни – опанування здобувачами методики аналізу та інспекції якості коду, концепцій рефакторингу коду, набуття практичних навичок застосування прийомів рефакторингу та інструментальних засобів для підвищення якості коду (табл. 2.1).

Таблиця 2.1

Теми лекційних занять та розподіл годин

№ з/п	Назва теми	год. (денна ф/н)	год. (заочна ф/н)
Тема 1	Аналіз та підвищення якості коду. Інструментальна база інспекції якості коду. Основні принципи та практики рефакторингу. Антипатерни як шаблони типових помилок. Поняття «чистий» та «брудний» код. Ознаки брудного коду.	4	2
Тема 2	Застосування Visual Studio для аналізу та рефакторингу коду. Прийоми рефакторингу: складання методів, переміщення функцій між об'єктами, організація даних, спрощення умовних виразів та викликів методів, задачі узагальнення об'єктів.	4	1
Тема 3	Запахи коду як індикатори глибинних проблем, їх пошук та прийоми виправлення: роздувальщики, порушники об'єктного дизайну, ускладнювачі змін, забрюднювачі коду, заплутувальники зв'язками.	4	1

Силабус навчальної дисципліни «Технології безперервного розроблення та інтеграції програмного забезпечення» освітньої програми «Інтегровані інформаційні системи» Національного технічного університету «Київський політехнічний інститут імені Ігоря Сікорського» [2] своєю метою визначає Підготовка висококваліфікованих фахівців, які володіють основними підходами до організації розробки програмних систем, проектування структури програмної системи, підвищення якості програмного коду, та забезпечення більш швидкої доставки в «продакшн» нового функціоналу.

Після проходження дисципліни студенти повинні вміти використовувати патерни в розробці програмного коду, вміти виконувати рефакторинг, знати процеси безперервної інтеграції програмного коду та безперервної доставки ПЗ.

Таблиця 2.2

Теми лекційних занять та розподіл годин

№ з/п	Назва теми	год. (денна ф/н)	год. (заочна ф/н)
Тема 1	Принципи рефакторингу. Признаки необхідності проведення рефакторингу. Розробка тестів. Рефакторинг та проектування. Рефакторинг та продуктивність.	4	1
Тема 2	Поняття запаху коду. Дублювання коду. Довгий метод. Коментарі. Великий клас. Довгий список параметрів. Групи даних. Оператори типу switch. Паралельні ієрархії наслідування. Лінивий клас. Тимчасове поле.	4	1
Тема 3	Демонстрація аналізу запахів коду та проведення рефакторингу на реальному проекті в Visual Studio. Демонстрація запахів коду в проекті, озвучення методів рефакторингу які будуть виконуватися, проведення озвучених методів рефакторингу, перевірка тестів що код залишається робочим.	2	1
Тема 4	Рефакторинги роботи з методами. Виділення методу. Inline Method. Заміна тимчасової змінної викликом методу. Введення пояснюючої змінної. Розщеплення пояснюючої змінної. Видалення присвоювань параметрам. Заміна метода об’єктом. Заміщення алгоритму.	4	2
Тема 5	Спрощення умовних виразів. Декомпозиція умовного оператора. Консолідація умовного виразу. Консолідація дублюючих умовних фрагментів. Видалення контрольного прапорця. Заміна вкладених умовних операторів граничним. Введення об’єкта Null. Введення твердження	4	2

Силабус навчальної дисципліни «Рефакторинг програмного забезпечення» ОПП «Комп’ютерні науки» Харківського національного університету радіоелектроніки [3] своєю метою визначає опанування студентами здатності знаходити та визначати частини програмного коду, які потребують застосування методів рефакторингу; розробляти тести програмного забезпечення; застосовувати переміщення функцій між об'єктами, організацію даних, реорганізацію коду, спрощення умовних виразів та викликів методів тощо; розв’язувати задачі узагальнення; застосовувати інструментальні засоби рефакторингу (табл. 2.3).

Таблиця 2.3

Теми лекційних занять та розподіл годин

№ з/п	Назва теми	год. (денна ф/н)	год. (заочна ф/н)
Тема 1	Введення. Принципи рефакторингу. Загальний огляд методів рефакторингу.	2	1
Тема 2	Пошук та визначення частин програмного коду, які потребують застосування методів рефакторингу. Розробка тестів програмного забезпечення.	2	1
Тема 3	Методи рефакторингу. Великі рефакторинги.	2	1
Тема 4	Повторне використання рефакторингу. Рефакторінг як засіб отримання вигод.	4	1
Тема 5	Безпека рефакторингу. Рефакторінг на практиці.	2	1
Тема 6	Рефакторінг з використанням інструментальних засобів	4	2
Тема 7	Практика рефакторингу у великих проектах.	4	2

Силабус навчальної дисципліни «Паттерни проєктування» ОПП «Інтелектуальний аналіз даних в комп’ютерних інформаційних системах» Чернівецького національного університету ім. Ю. Федьковича [4] своєю метою визначає формування у студентів представлень про способи проєктування програмного забезпечення на основі моделі предметної області, отримання вмінь ефективного застосування компонентів багаторазового використання при використанні технології ООП.

Студенти повинні опанувати практичні аспекти використання патернів проєктування у задачах створення програмних застосунків різного типу

складності, кодові запахи, принципи та задачі рефакторингу, роль gof-патернів у задачах рефакторингу (табл. 2.4).

Таблиця 2.4

Теми лекційних занять та розподіл годин

№ з/п	Назва теми	год. (денна ф/н)	год. (заочна ф/н)
Тема 1	SOLID принципи. S: Single Responsibility Principle (SRP), O: Open closed Principle (OCP), L: Liskov substitution Principle (LSP), I: Interface Segregation Principle (ISP), D: Dependency Inversion Principle (DIP)	4	1
Тема 2	Запахи коду. Роздувальники. Порушники об'єктного дизайну. Ускладнювачі змін. Забрюднювачі коду. Заплутувальники зв'язками.	4	1
Тема 3	Техніки рефакторингу. Складання методів. Переміщення функцій між об'єктами. Організація даних. Спрощення умовних виразів. Спрощення викликів методів. Задачі узагальнення об'єктів	4	1

2.2 Постановка лабораторної роботи «Група прийомів рефакторингу “Складання методів”»

Постановка завдання лабораторної роботи.

1. Опрацюйте теоретичний матеріал: ознайомтесь з специфікою групи прийомів рефакторингу «Складання методів».

2. Виконати проєктування та програмну реалізацію предметної області, обраної за варіантом. Вважати програмну реалізацію брудним кодом.

3. Виконати рефакторинг фрагментів коду програмної реалізації предметної області. Необхідно застосувати наступний перелік прийомів рефакторингу групи «Складання методів»: відокремлення методу, вбудовування методу, відокремлення змінної, заміна змінної викликом методу, видалення присвоювань параметрам. Обов’язково написати коментарі до ключових етапів.

4. Побудувати UML-діаграму ієрархії класів відповідно фрагментам брудного та чистого коду мовою PlantUML у сервісі Draw.io.

5. Підготувати звіт до захисту лабораторної роботи:

– опис предметної області;

– UML-діаграма ієрархії класів брудного та чистого кодів;

– фрагменти брудного та чистого кодів мовою програмування C#, C++, TypeScript або Java;

– виконати опис змін в коді після прийому рефакторингу;

– в оформленні коду дотримуватись єдиного стилю та доповнювати код коментарями.

Етапи виконання лабораторної роботи.

Опис предметної області, обраної на індивідуальним інваріантом: розробити клас Produce, який містить назву продукту та ціну одиниці продукту, визначити конструктори, деструктор та методи встановлення і читання значень полів даних. Визначити похідний клас Ingredient з додатковими даними про дозування продукту. Визначити операторні методи введення, виведення, корегування полів даних класу. Розробити клас Recipe кулінарного виробу, який містить масив об’єктів класу Ingredient.

1. Прийом відокремлення методу (Extract Method). Проблема: у вас є фрагмент коду, який можна згрупувати. Рішення: виділіть цей фрагмент в новий метод (чи функцію) і викличте його замість старого коду.

Фрагмент брудного коду мовою C#:

```
public override string ToString()
{
    // Ініціалізація рядка з заголовком
    string recipeInfo = " Інгредієнти рецепта:\n";

    // Проходимося по кожному інгредієнту в списку
    foreach (var ingredient in Ingredients)
    {
        // Додаємо інформацію про інгредієнт до рядка
        recipeInfo += ingredient.ToString() + "\n";
    }
    // Додаємо загальну вартість рецепту
    recipeInfo += $" Загальна вартість: {CalculateTotalCost():C}";
    return recipeInfo; // Повертаємо сформований рядок
}
```

Фрагмент чистого коду мовою C#:

```
public override string ToString()
{
    string recipeInfo = GenerateRecipeInfo();
    // Викликаємо новий метод для отримання інформації про інгредієнти
    recipeInfo += $" Загальна вартість: {CalculateTotalCost():C}";
    // Додаємо загальну вартість рецепту
```

```
    return recipeInfo; // Повертаємо сформований рядок
}
private string GenerateRecipeInfo()
{
    // Ініціалізація рядка з заголовком
    string info = " Інгредієнти рецепта:\n";
    foreach (var ingredient in Ingredients)
    // Проходимося по кожному інгредієнту в списку
    {
        info += ingredient.ToString() + "\n";
        // Додаємо інформацію про інгредієнт до рядка
    }
    return info; // Повертаємо сформований рядок з інформацією про інгредієнти
}
```

Рисунок 2.1 – Фрагменти UML-діаграм до та після прийому відокремлення методу

Опис змін в коді після прийому рефакторингу: створено новий метод GenerateRecipeInfo() – вся логіка, пов'язана з формуванням інформації про інгредієнти, винесена в окремий метод. Це підвищує читабельність коду, оскільки ToString() став більш зрозумілим. Окремий метод дозволяє повторно використовувати код, якщо в майбутньому знадобиться отримати інформацію про інгредієнти в інших частинах програми. Код став більш структурованим і легким для розуміння. Тепер основна логіка ToString() полягає в тому, щоб викликати два методи: один для отримання списку інгредієнтів, інший — для розрахунку загальної вартості.

2. Прийом вбудовування методу (Inline Method). Проблема: варто використовувати у випадках, коли тіло методу очевидніше за сам метод. Рішення: замініть виклики методу його вмістом і видаліть сам метод.

Фрагмент брудного коду мовою C#:

```
public decimal CalculateCost()
{
    // Обчислюємо вартість інгредієнта
    return PricePerUnit * (decimal)Dosage;
}
```

Фрагмент чистого коду мовою C#:

```
public decimal Cost
```

```
{
    // Обчислюємо вартість інгредієнта через властивість
    get { return PricePerUnit * (decimal)Dosage; }
}
```

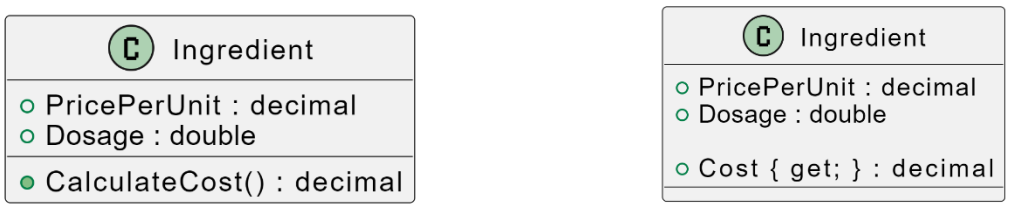

Рисунок 2.2 – Фрагменти UML-діаграм до та після прийому вбудовування методу

Опис змін в коді після прийому рефакторингу: метод CalculateCost() має досить просту логіку і завжди повертає обчислене значення на основі властивостей класу. Виклик методу для отримання вартості може бути зайвим, оскільки це не є функціональною логікою. Після рефакторингу метод CalculateCost() перетворено на властивість Cost, що дозволяє читати вартість як звичайне поле. Тепер користувач класу може просто звернутися до Cost, не замислюючись про виклик методу, що робить код зрозумілішим і чистішим. Використання властивості зменшує складність коду, оскільки прості обчислення тепер мають простий синтаксис для доступу.

3. Прийом відокремлення змінної (Extract Variable). Проблема: у вас є складний для розуміння вираз. Рішення: помістіть результат виразу або його частини в окремі змінні, що пояснюють суть виразу.

Фрагмент брудного коду мовою C#:

```
// Обчислюємо загальну вартість рецепту
      public decimal TotalCost => Ingredients.Sum(ingredient => ingredient.Cost);
```

Фрагмент чистого коду мовою C#:

```
public decimal TotalCost
{
    get
    {
        // Винесення результату в окрему змінну
        decimal totalCost = Ingredients.Sum(ingredient => ingredient.Cost);
        return totalCost; // Повертаємо загальну вартість рецепту
    }
}
```

Опис змін в коді після прийому рефакторингу: введення змінної totalCost дозволяє зберігати обчислення, що покращує ясність коду. Тепер код легше читати, і можна швидше зрозуміти, що TotalCost повертає загальну вартість, обчислену за допомогою списку інгредієнтів. В майбутньому можна легко додати додаткову обробку або перевірки, не змінюючи структуру коду.

4. Прийом заміна змінної викликом методу (Replace Temp with Query). Проблема: ви розміщуєте результат якогось виразу в локальній змінній, щоб використати її далі в коді. Рішення: виділіть цей вираз в окремий метод і повертайте результат з нього. Замініть використання вашої змінної викликом методу. Новий метод може бути використаний і в інших методах.

Фрагмент брудного коду мовою C#:

```
public decimal CalculateTotalCost()
{
    // Зберігаємо результат обчислення в змінну
    decimal totalCost = Ingredients.Sum(ingredient => ingredient.Cost);
    return totalCost; // Повертаємо загальну вартість рецепту
}
```

Фрагмент чистого коду мовою C#:

```
// Викликаємо метод для отримання загальної вартості
public decimal TotalCost => CalculateTotalCost();

private decimal CalculateTotalCost()
{
    // Повертаємо суму вартостей інгредієнтів
    return Ingredients.Sum(ingredient => ingredient.Cost);
}
```

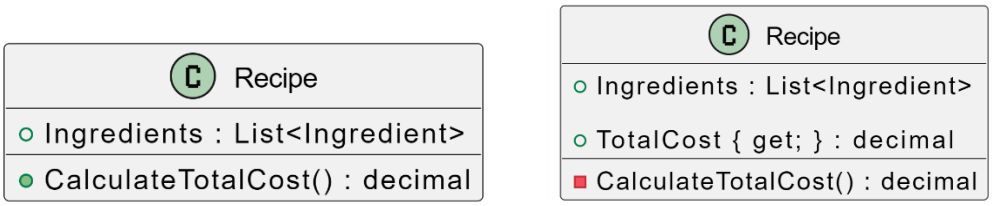

Рисунок 2.3 – Фрагменти UML-діаграм до та після прийому заміна змінної викликом методу

Опис змін в коді після прийому рефакторингу: замість зберігання значення в змінній, використовується властивість TotalCost, яка викликає метод CalculateTotalCost(). Логіка, що обчислює загальну вартість, винесена в приватний метод, що дозволяє повторно використовувати цю логіку в інших місцях класу, якщо це знадобиться. Виділення логіки обчислення в окремий

метод дозволяє легко змінювати, тестувати чи вдосконалювати цю логіку в майбутньому без зміни самого класу.

5. Прийом видалення присвоювань параметрам (Remove Assignments to Parameters). Проблема: параметру методу присвоюється якесь значення. Рішення: замість параметра скористайтеся новою локальною змінною.

Фрагмент брудного коду мовою C#:

```
public void AddIngredient(Ingredient ingredient)
{
    if (ingredient == null) // Перевіряємо, чи параметр ingredient є null
    {
        // Присвоюємо новий об'єкт, якщо ingredient був null
        ingredient = new Ingredient("За замовченням", 0m, 0);
    }
    Ingredients.Add(ingredient); // Додаємо інгредієнт до списку
}
```

Фрагмент чистого коду мовою C#:

```
public void AddIngredient(Ingredient ingredient)
{
    // Створюємо нову локальну змінну і присвоюємо їй значення параметра
    Ingredient ingredientToAdd = ingredient;
    if (ingredientToAdd == null) // Перевіряємо, чи локальна змінна є null
    {
        // Присвоюємо новий об'єкт, якщо ingredientToAdd був null
        ingredientToAdd = new Ingredient("За замовченням", 0m, 0);
    }
    // Додаємо інгредієнт до списку
    Ingredients.Add(ingredientToAdd);
}
```

Опис змін в коді після прийому рефакторингу: присвоєння нового значення параметру ingredient може ввести в оману, адже це не є звичним для методів, саме тому замість того, щоб змінювати параметр, створюється нова змінна ingredientToAdd, що зберігає його значення. Тепер чітко видно, що метод не змінює вхідний параметр, а працює з новим значенням. Локальні змінні полегшують тестування, оскільки їх використання не впливає на зовнішній стан.

2.3 Постановка лабораторної роботи «Група прийомів рефакторингу “Переміщення функцій між об'єктами”»

Постановка завдання лабораторної роботи.

1. Опрацюйте теоретичний матеріал: ознайомтесь з специфікою групи

прийомів рефакторингу «Переміщення функцій між об'єктами».

2. Виконати проєктування та програмну реалізацію предметної області, обраної за індивідуальним варіантом. Вважати програмну реалізацію брудним кодом.

3. Виконати рефакторинг фрагментів коду програмної реалізації предметної області. Необхідно застосувати наступний перелік прийомів рефакторингу групи «Переміщення функцій між об'єктами»: переміщення методу, вбудовування класу, приховання делегування, введення локального розширення, введення зовнішнього методу. Обов’язково написати коментарі до ключових етапів.

4. Побудувати UML-діаграму ієрархії класів відповідно фрагментам брудного та чистого коду мовою PlantUML у сервісі Draw.io.

5. Підготувати звіт до захисту лабораторної роботи, який повинен містити:

– опис предметної області;

– UML-діаграма ієрархії класів брудного та чистого кодів;

– фрагменти брудного та чистого кодів мовою програмування C#, C++, TypeScript або Java;

– виконати опис змін в коді після прийому рефакторингу;

– в оформленні коду дотримуватись єдиного стилю та доповнювати код коментарями.

Етапи виконання лабораторної роботи.

1. Прийом переміщення методу (Move Method). Проблема: метод використовується в іншому класі більше, ніж у власному. Рішення: створіть новий метод в класі, який використовує його більше інших, і перенесіть туди код із старого методу. Код оригінального методу перетворіть на звернення до нового методу в іншому класі або приберіть його взагалі.

Фрагмент брудного коду мовою C#:

```
public class BankAccount
{
    public string AccountNumber { get; private set; }
    public decimal Balance { get; private set; }
```

```
    public BankAccount(string accountNumber, decimal balance)
    {
        AccountNumber = accountNumber;
        Balance = balance;
    }

    public void Deposit(decimal amount)
    {
        if (amount > 0)
        {
            Balance += amount;
        }
    }

    public void Withdraw(decimal amount)
    {
        if (amount > 0 && amount <= Balance)
        {
            Balance -= amount;
        }
    }

    public bool IsOverdraft()
    {
        return Balance < 0;
    }
}

public class Customer
{
    public string Name { get; private set; }
    public BankAccount Account { get; private set; }

    public Customer(string name, BankAccount account)
    {
        Name = name;
        Account = account;
    }

    public void TransferTo(BankAccount destinationAccount, decimal amount)
    {
        if (amount > 0 && Account.Balance >= amount)
        {
            Account.Withdraw(amount);
            destinationAccount.Deposit(amount);
        }
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class BankAccount
{
    public string AccountNumber { get; private set; }
    public decimal Balance { get; private set; }

    public BankAccount(string accountNumber, decimal balance)
    {
        AccountNumber = accountNumber;
        Balance = balance;
    }

    public void Deposit(decimal amount)
    {
        if (amount > 0)
```

```
        {
            Balance += amount;
        }
    }

    public void Withdraw(decimal amount)
    {
        if (amount > 0 && amount <= Balance)
        {
            Balance -= amount;
        }
    }

    public bool IsOverdraft()
    {
        return Balance < 0;
    }

    public void TransferTo(BankAccount destinationAccount, decimal amount)
    {
        if (amount > 0 && Balance >= amount)
        {
            Withdraw(amount);
            destinationAccount.Deposit(amount);
        }
    }
}

public class Customer
{
    public string Name { get; private set; }
    public BankAccount Account { get; private set; }

    public Customer(string name, BankAccount account)
    {
        Name = name;
        Account = account;
    }
}
```

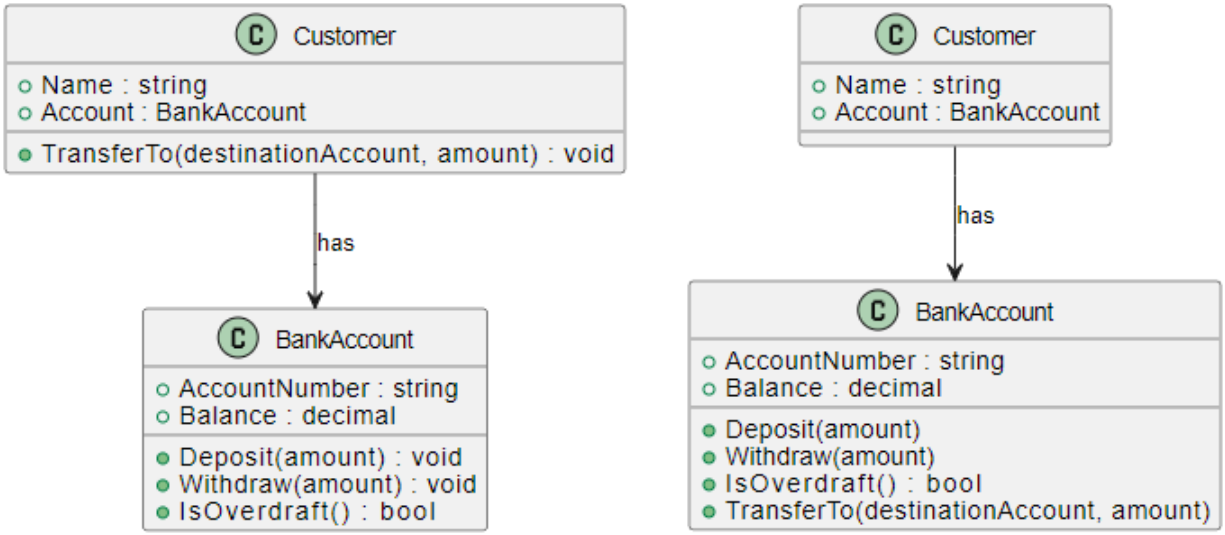

Рисунок 2.4 – Фрагменти UML-діаграм до та після переміщення методу

Опис змін в коді після прийому рефакторингу: метод TransferTo у

брудному коді був розміщений в класі Customer, хоча цей метод більше відповідає функціоналу класу BankAccount, який відповідає за операції з рахунком. Після виконання прийому рефакторингу метод TransferTo переміщений з класу Customer до класу BankAccount, де він більше відповідає функціоналу обробки операцій з рахунком. Цей рефакторинг робить код більш чистим і спрощує логіку кожного класу.

2. Прийом вбудовування класу (Inline Class). Проблема: клас майже нічого не робить, ні за що не відповідає, і ніякої відповідальності для цього класу не планується. Рішення: перемістіть функціонал з цього класу в інший. Перевага прийому: менше даремних класів – більше вільної оперативної пам’яті, у тому числі, і у вас в пам’яті.

Фрагмент брудного коду мовою C#:

```
public class Address
{
    public string Street { get; set; }
    public string City { get; set; }
    public string ZipCode { get; set; }
}

public class Customer
{
    public string Name { get; set; }
    public Address Address { get; set; }
}

public class Order
{
    public int OrderNumber { get; set; }
    public Customer Customer { get; set; }
    public Address ShippingAddress { get; set; }
    public Address BillingAddress { get; set; }

    public decimal CalculateTotalPrice()
    {
        // Розрахунок загальної вартості замовлення.
        // ...
        return 0;
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Customer
{
    public string Name { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string ZipCode { get; set; }
}

public class Order
```

```
{
    public int OrderNumber { get; set; }
    public string CustomerName { get; set; }
    public string ShippingStreet { get; set; }
    public string ShippingCity { get; set; }
    public string ShippingZipCode { get; set; }
    public string BillingStreet { get; set; }
    public string BillingCity { get; set; }
    public string BillingZipCode { get; set; }

    public decimal CalculateTotalPrice()
    {
        // Розрахунок загальної вартості замовлення.
        // ...
        return 0;
    }
}
```

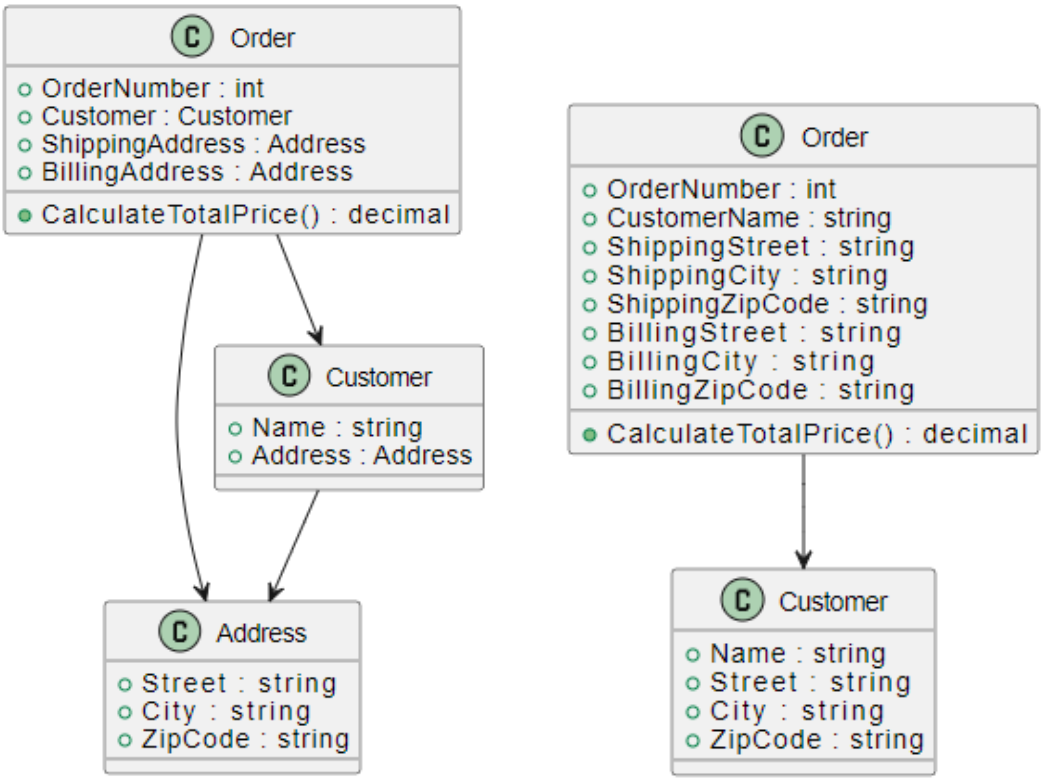

Рисунок 2.5 – Фрагменти UML-діаграм до та після вбудовування класу

Опис змін в коді після прийому рефакторингу: у брудному коді клас Address має дуже обмежений функціонал і використовується як частина інших класів, таких як Customer і Order. Після рефакторингу властивості класу Address були безпечно перенесені в класи Customer і Order. Це робить код менш складним і спрощує структуру даних. Прийом вбудовування класу може бути застосований там, де клас має обмежений функціонал і не має власної суттєвої логіки.

3. Прийом приховання делегування (Hide Delegate). Проблема: клієнт

отримує об’єкт B з поля або методу об’єкта A. Потім клієнт викликає якийсь метод об’єкта B. Рішення: створіть новий метод в класі A, який би делегував виклик об’єкта B. Таким чином, клієнт перестане знати про клас B і залежати від нього.

Фрагмент брудного коду мовою C#:

```
public class Department
{
    public string Name { get; set; }
    public Employee Manager { get; set; }
}

public class Employee
{
    public string Name { get; set; }
    public Department Department { get; set; }

    public Employee GetManager()
    {
        return Department.Manager;
    }
}

public class Client
{
    public void PrintManagerName(Employee employee)
    {
        var manager = employee.GetManager();
        Console.WriteLine("Manager: " + manager.Name);
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Department
{
    public string Name { get; set; }
    public Employee Manager { get; set; }
}

public class Employee
{
    public string Name { get; set; }
    public Department Department { get; set; }

    public Employee GetManager()
    {
        return Department.Manager;
    }

    // Hide Delegate: Добавлено прямий метод для отримання керівника.
    public Employee GetDepartmentManager()
    {
        return GetManager();
    }
}

public class Client
{
    public void PrintManagerName(Employee employee)
    {
```

```
        var manager = employee.GetDepartmentManager();
        Console.WriteLine("Manager: " + manager.Name);
    }
}
```

Опис змін в коді після прийому рефакторингу: у брудному коді клас Client звертається до менеджера через метод GetManager() у класі Employee, який повертає Manager з об'єкта Department. Клієнтський клас змушений знати про існування об'єкта Department всередині Employee, що може ускладнювати підтримку коду та робити його більш вразливим до змін у класі Department. Після рефакторингу у класі Employee додано новий метод GetDepartmentManager(), який повертає менеджера напряму. Тепер клієнтський код не потребує знати про взаємозв'язок між Employee і Department. Це підвищує інкапсуляцію та зменшує зв'язаність класів.

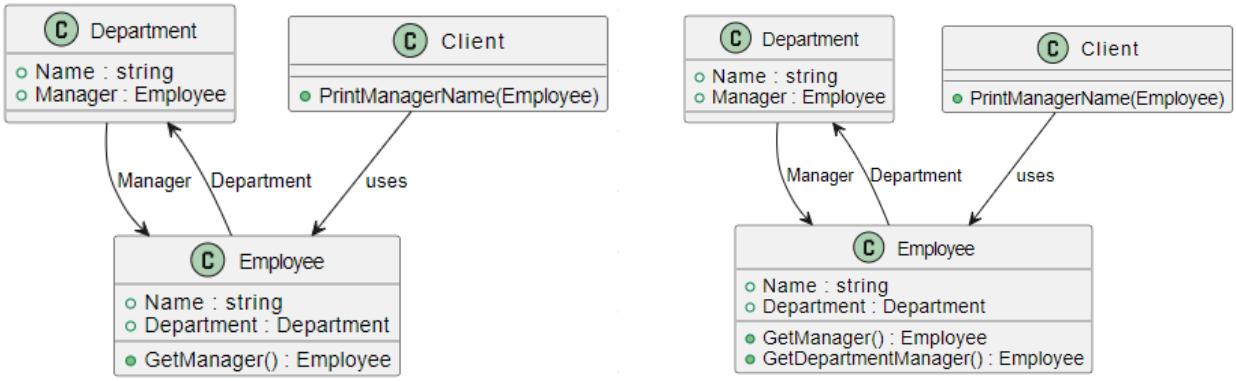

Рисунок 2.6 – Фрагменти UML-діаграм до та після приховання делегування

4. Прийом введення локального розширення (Introduce Local Extension). Проблема: в службовому класі відсутні деякі методи, які вам потрібні. При цьому додати їх в цей клас ви не можете. Рішення: створіть новий клас, який би містив ці методи, і зробіть його спадкоємцем службового класу, або його обгорткою.

Фрагмент брудного коду мовою C#:

```
public class Medication
{
    public string Name { get; set; }
    public double Price { get; set; }
}
public class Program
{
    public static void Main()
    {
        Medication medication1 = new Medication { Name = "Аспірин", Price = 75 };
        Medication medication2 = new Medication { Name = "Ібупрофен", Price = 120 };
```

```
        // саме для цієї логіки можна створити розширення до класу Medication,
        // яке буде обчислювати, чи вистачить грошей на ліки чи ні;
        bool isMed1Affordable = medication1.Price < 100;
        bool isMed2Affordable = medication2.Price < 100;

        Console.WriteLine("Чи вистачає нам грошей на перші ліки? "
            + isMed1Affordable);
        Console.WriteLine("Чи вистачає нам грошей на другі ліки? "
            + isMed2Affordable);
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Medication
{
    public string Name { get; set; }
    public double Price { get; set; }
}

// Для цієї логіки ми створили створили розширення до класу Medication:
// MedicationExtensions, яке буде обчислювати, чи вистачить грошей на ліки чи ні;
public static class MedicationExtensions
{
    // Визначимо метод розширення, який оперуватиме об'єктами Medication.
    public static bool IsAffordable(this Medication medication)
    {
        return medication.Price < 100;
    }
}
public class Program
{
    public static void Main()
    {
        Medication medication1 = new Medication { Name = "Аспірин", Price = 75 };
        Medication medication2 = new Medication { Name = "Ібупрофен", Price = 120 };

        bool isMed1Affordable = medication1.IsAffordable();
        bool isMed2Affordable = medication2.IsAffordable();

        Console.WriteLine("Чи вистачає нам грошей на перші ліки? "
            + isMed1Affordable);
        Console.WriteLine("Чи вистачає нам грошей на другі ліки? "
            + isMed2Affordable);
    }
}
```

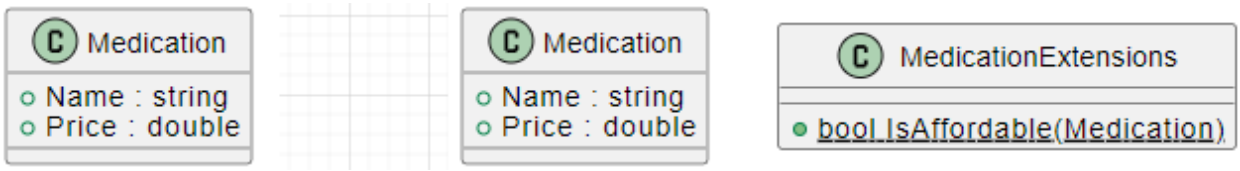

Рисунок 2.7 – Фрагменти UML-діаграм до та після введення локального розширення

Опис змін в коді після прийому рефакторингу: у чистому коді додано новий статичний клас MedicationExtensions, що містить метод розширення

IsAffordable. Це дозволяє реалізувати логіку доступності медикаментів у відокремленому місці, що підвищує модульність і повторне використання коду. У класі Program логіка перевірки доступності медикаментів перенесена з основного методу Main у метод IsAffordable. Це усуває повторення коду та підвищує його читабельність. Тепер перевірка доступності виконується простим викликом методу medication1.IsAffordable() замість порівняння цін безпосередньо.

5. Прийом введення зовнішнього методу (Introduce Foreign Method). Проблема: службовий клас (який не контролюється вами) не містить методу, який вам потрібен, при цьому у вас немає можливості додати метод в цей клас. Рішення: додайте метод в клієнтський клас і далі передаватиме в нього об’єкт службового класу в якості аргументу.

Фрагмент брудного коду мовою C#:

```
public class User
{
    public string Name { get; set; }
}
public class UserService
{
    public void AddUser(User user)
    {
        // Логіка для додавання користувача
        Console.WriteLine($"User {user.Name} added.");
    }
}

public class UserManager
{
    private UserService userService;

    public UserManager(UserService userService)
    {
        this.userService = userService;
    }

    public void AddUserWithLogging(User user)
    {
        // Немає можливості додати логіку в UserService
        userService.AddUser(user);
        Console.WriteLine($"User {user.Name} added at {DateTime.Now}.");
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class User
{
    public string Name { get; set; }
}
public class UserService
```

```
{
    public void AddUser(User user)
    {
        // Логіка для додавання користувача
        Console.WriteLine($"User {user.Name} added.");
    }
}
public class UserManager
{
    private UserService userService;

    public UserManager(UserService userService)
    {
        this.userService = userService;
    }

    public void AddUserWithLogging(User user)
    {
        AddUserAndLog(userService, user);
    }

    private void AddUserAndLog(UserService userService, User user)
    {
        userService.AddUser(user);
        Console.WriteLine($"User {user.Name} added at {DateTime.Now}.");
    }
}
```

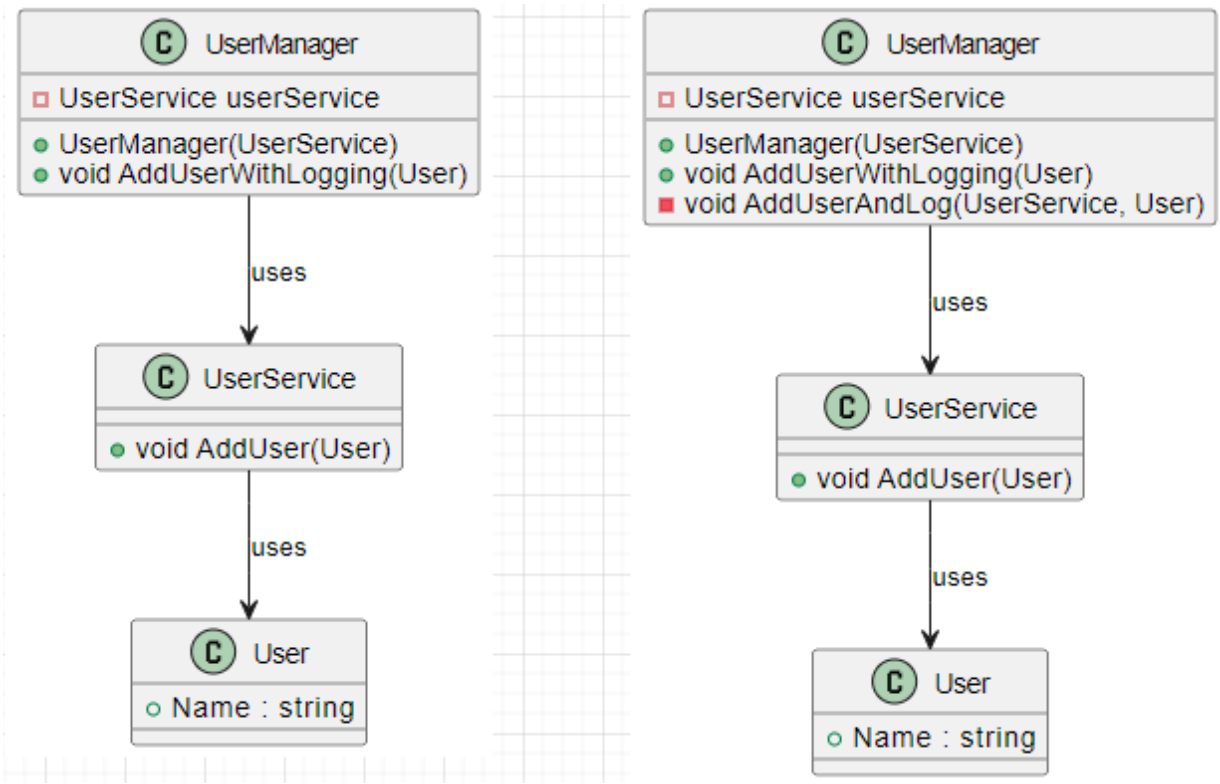

Рисунок 2.8 – Фрагменти UML-діаграм до та після введення зовнішнього методу

Опис змін в коді після прийому рефакторингу: у брудному коді логіка для додавання користувача та логування додавання були реалізовані в одному

методі AddUserWithLogging класу UserManager. У чистому коді створено приватний метод AddUserAndLog, який відповідає за додавання користувача та логування. Рефакторинг дозволив зменшити дублювання логіки, оскільки тепер логіка для додавання користувача і логування винесена в окремий метод, що робить код більш зрозумілим і зручним для подальшої підтримки.

2.4 Постановка лабораторної роботи «Група прийомів рефакторингу “Організація даних”»

Постановка завдання лабораторної роботи.

1. Опрацюйте теоретичний матеріал: ознайомтесь з специфікою групи прийомів рефакторингу «Організація даних».

2. Виконати проєктування та програмну реалізацію предметної області, обраної за індивідуальним варіантом. Вважати програмну реалізацію брудним кодом.

3. Виконати рефакторинг фрагментів коду програмної реалізації предметної області. Необхідно застосувати наступний перелік прийомів рефакторингу групи «Організація даних»: самоінкапсуляція поля, інкапсуляція колекції, заміна підкласу полями, заміна поля-масиву об'єктом, заміна двонаправленого зв'язку одностороннім.

5. В брудному та чистому коді обов’язково вказати клас Program з головною точкою входу static void Main().

6. Побудувати UML-діаграму ієрархії класів відповідно фрагментам брудного та чистого коду мовою PlantUML у сервісі Draw.io.

7. Підготувати звіт до захисту лабораторної роботи, який повинен містити:

– опис предметної області;

– UML-діаграма ієрархії класів брудного та чистого кодів;

– фрагменти брудного та чистого кодів мовою програмування C#, C++, TypeScript або Java;

– виконати опис змін в коді після прийому рефакторингу;

– в оформленні коду дотримуватись єдиного стилю та доповнювати код коментарями.

Етапи виконання лабораторної роботи.

1. Прийом самоінкапсуляція поля (Self Encapsulate Field). Проблема: ви використовуєте прямий доступ до приватних полів усередині класу. Рішення: створіть методу доступу Get() і Set() для поля, і користуйтеся для доступу до поля тільки ними.

Фрагмент брудного коду мовою C#:

```
public class Produce
{
    public string name; // відкрите поле для назви продукту
    public double unitPrice; // відкрите поле для ціни одиниці продукту

    public Produce(string name, double unitPrice)
    {
        this.name = name; // встановлення назви продукту
        this.unitPrice = unitPrice; // встановлення ціни одиниці продукту
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Produce
{
    private string name; // приватне поле для назви продукту
    private double unitPrice; // приватне поле для ціни одиниці продукту
    public Produce(string name, double unitPrice)
    {
        SetName(name); // метод для встановлення назви продукту
        SetUnitPrice(unitPrice); // метод для встановлення ціни одиниці продукту
    }
    // метод для отримання назви продукту
    public string GetName()
    {
        return name;
    }
    // метод для встановлення назви продукту
    public void SetName(string name)
    {
        this.name = name;
    }
    // метод для отримання ціни одиниці продукту
    public double GetUnitPrice()
    {
        return unitPrice;
    }
    // метод для встановлення ціни одиниці продукту
    public void SetUnitPrice(double unitPrice)
    {
        this.unitPrice = unitPrice;
    }
}
```

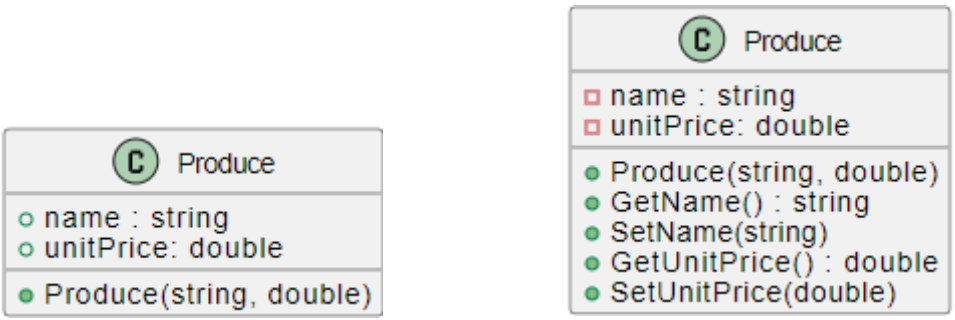

Рисунок 2.9 – Фрагменти UML-діаграм до та після прийому самоінкапсуляція поля

Опис змін в коді після прийому рефакторингу: у брудному коді поля name та unitPrice є публічними, що дозволяє змінювати їх з будь-якої частини програми. Це порушує принцип інкапсуляції, бо зовнішній код може неконтрольовано модифікувати поля об'єкта. Після рефакторингу поля стали приватними, і доступ до них відбувається через публічні методи доступу (гетери і сетери). Це забезпечує контроль над тим, як і коли змінюються поля. У майбутньому можна легко додати перевірки або інші дії при встановленні чи отриманні значень полів.

2. Прийом інкапсуляція колекції (Encapsulate Collection). Клас містить поле-колекцію, а також простий Get() і Set() для роботи з цією колекцією. Рішення: Зробіть значення, що повертає Get(), доступним тільки для читання, а також створіть методи додавання/видалення елементів цієї колекції.

Фрагмент брудного коду мовою C#:

```
public class Recipe
{
    public Ingredient[] ingredients; // публічний масив інгредієнтів

    public Recipe(Ingredient[] ingredients)
    {
        this.ingredients = ingredients; // встановлення масиву інгредієнтів
    }

    public double CalculateCost()
    {
        double totalCost = 0;
        foreach (Ingredient ingredient in ingredients)
        {
            // обчислення загальної вартості
            totalCost += ingredient.unitPrice * ingredient.dosage;
        }
        return totalCost;
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Recipe
{
    private List<Ingredient> ingredients; // приватний список інгредієнтів

    public Recipe()
    {
        ingredients = new List<Ingredient>(); // ініціалізація порожнього списку
    }

    // метод для додавання інгредієнта
    public void AddIngredient(Ingredient ingredient)
    {
        ingredients.Add(ingredient);
    }
    // метод для видалення інгредієнта
    public void RemoveIngredient(Ingredient ingredient)
    {
        ingredients.Remove(ingredient);
    }

    // метод для отримання колекції інгредієнтів, доступної тільки для читання
    public IEnumerable<Ingredient> GetIngredients()
    {
        return ingredients.AsReadOnly();
    }

    public double CalculateCost()
    {
        double totalCost = 0;
        foreach (Ingredient ingredient in ingredients)
        {
            // обчислення загальної вартості
            totalCost += ingredient.GetUnitPrice() * ingredient.GetDosage();
        }
        return totalCost;
    }
}
```

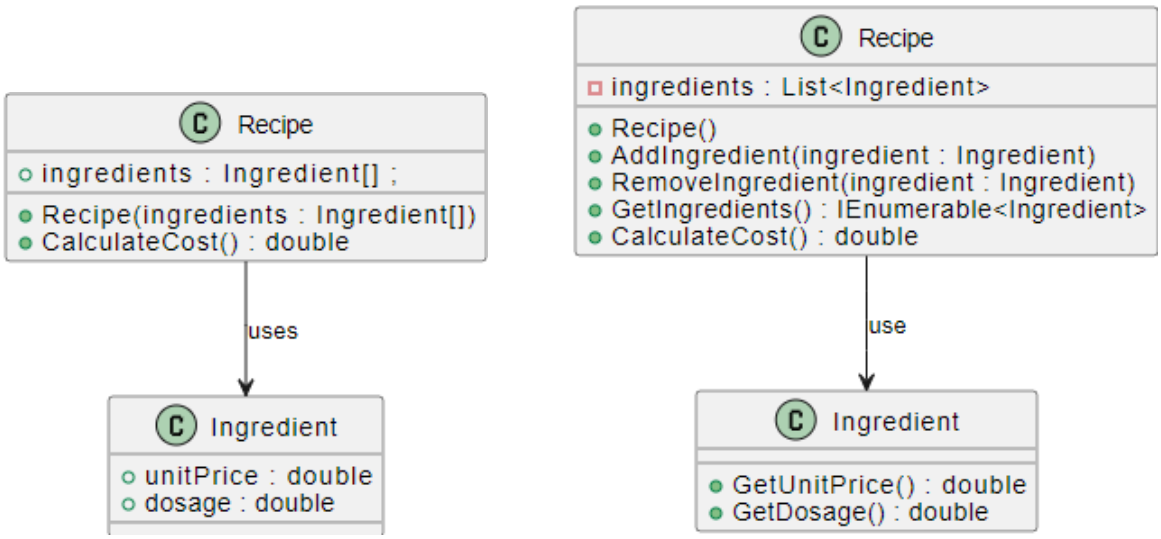

Рисунок 2.10 – Фрагменти UML-діаграм до та після прийому інкапсуляція колекції

Опис змін в коді після прийому рефакторингу: у брудному коді масив інгредієнтів ingredients був публічним, що надавало зовнішнім об'єктам прямий доступ до колекції. Це порушувало принцип інкапсуляції, оскільки зовнішні об'єкти могли змінювати колекцію без контролю з боку класу. У чистому коді колекція ingredients стала приватною, доступною лише через спеціальні методи класу. Це дає змогу контролювати, як і коли колекція змінюється, забезпечуючи краще управління її станом. Також додано методи для роботи з колекцією AddIngredient та RemoveIngredient. Вони контролюють додавання та видалення елементів у списку, забезпечуючи безпеку операцій. Це дозволяє зберігати колекцію в послідовному стані та уникати прямого доступу до внутрішньої структури класу.

3. Прийом заміна підкласу полями (Replace Subclass with Fields). Проблема: у вас є підкласи, які відрізняються тільки методами, що повертають дані-константи. Рішення: замініть методи полями в батьківському класі і видаліть підкласи.

Фрагмент брудного коду мовою C#:

```
public class Produce
{
    public string name; // відкрите поле для назви продукту
    public double unitPrice; // відкрите поле для ціни одиниці продукту

    public Produce(string name, double unitPrice)
    {
        this.name = name; // встановлення назви продукту
        this.unitPrice = unitPrice; // встановлення ціни одиниці продукту
    }
}

public class Ingredient : Produce
{
    public double dosage; // поле для дозування

    public Ingredient(string name, double unitPrice, double dosage)
        : base(name, unitPrice)
    {
        this.dosage = dosage; // встановлення дозування
    }
    public void SetDosage(double dosage)
    {
        this.dosage = dosage; // метод для встановлення дозування
    }
    public double GetDosage()
    {
        return dosage; // метод для отримання дозування
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Produce
{
    private string name; // приватне поле для назви продукту
    private double unitPrice; // приватне поле для ціни одиниці продукту
    private double dosage; // приватне поле для дозування продукту
    public Produce(string name, double unitPrice, double dosage)
    {
        SetName(name); // встановлення назви продукту через метод
        SetUnitPrice(unitPrice); // встановлення ціни одиниці продукту через метод
        SetDosage(dosage); // встановлення дозування продукту через метод
    }
    // метод для отримання назви продукту
    public string GetName()
    {
        return name;
    }
    // метод для встановлення назви продукту
    public void SetName(string name)
    {
        this.name = name;
    }
    // метод для отримання ціни одиниці продукту
    public double GetUnitPrice()
    {
        return unitPrice;
    }
    // метод для встановлення ціни одиниці продукту
    public void SetUnitPrice(double unitPrice)
    {
        this.unitPrice = unitPrice;
    }
    // метод для отримання дозування продукту
    public double GetDosage()
    {
        return dosage;
    }
    // метод для встановлення дозування продукту
    public void SetDosage(double dosage)
    {
        this.dosage = dosage;
    }
}
```

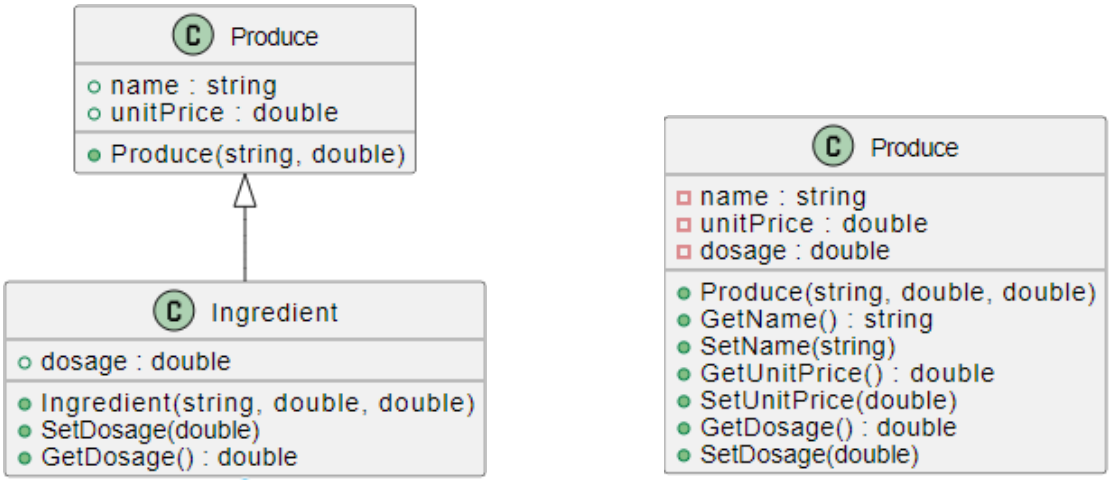

Рисунок 2.11 – Фрагменти UML-діаграм до та після прийому заміна підкласу полями

Опис змін в коді після прийому рефакторингу: у брудному коді клас Ingredient успадковував клас Produce, що створювало відокремлену ієрархію, в якій Ingredient є підкласом. У чистому коді вся інформація об'єднана в один клас Produce, що полегшує управління даними, оскільки всі поля тепер перебувають в одному класі. Поля name, unitPrice та dosage стали приватними. Також створені методи SetName, SetUnitPrice, і SetDosage, які використовуються для ініціалізації значень полів через конструктор. Це підвищує безпеку даних, оскільки доступ до них відбувається лише через публічні методи, що зменшує ризик змін зовнішніми класами.

4. Прийом заміна поля-масиву об'єктом (Replace Array with Object). Проблема: у вас є масив, в якому зберігаються різнотипні дані. Рішення: замініть масив об’єктом, який матиме окремі поля для кожного елементу.

Фрагмент брудного коду мовою C#:

```
public class Recipe
{
    private List<Ingredient> ingredients; // список інгредієнтів
    public Recipe()
    {
        ingredients = new List<Ingredient>(); // ініціалізація списку інгредієнтів
    }
    // метод для додавання інгредієнта
    public void AddIngredient(Ingredient ingredient)
    {
        ingredients.Add(ingredient);
    }
    // метод для обчислення загальної вартості
    public double CalculateCost()
    {
        double totalCost = 0;
        foreach (Ingredient ingredient in ingredients)
        {
            totalCost += ingredient.GetUnitPrice() * ingredient.GetDosage();
        }
        return totalCost;
    }
}
```

Фрагмент чистого коду мовою C#:

```
public class Ingredients
{
    private List<Ingredient> ingredientList; // список інгредієнтів
    public Ingredients()
    {
        // ініціалізація списку інгредієнтів
        ingredientList = new List<Ingredient>();
    }
    // метод для додавання інгредієнта
    public void Add(Ingredient ingredient)
    {
```

```
            ingredientList.Add(ingredient);
        }
        // метод для обчислення загальної вартості
        public double CalculateTotalCost()
        {
            double totalCost = 0;
            foreach (Ingredient ingredient in ingredientList)
            { totalCost += ingredient.GetUnitPrice() * ingredient.GetDosage(); }
            return totalCost;
        }
        // метод для отримання колекції інгредієнтів, доступної тільки для читання
        public IEnumerable<Ingredient> GetIngredients()
        {
            return ingredientList.AsReadOnly();
        }
    }

    public class Recipe
    {
        private Ingredients ingredients; // поле для управління інгредієнтами
        public Recipe()
        {
            ingredients = new Ingredients(); // ініціалізація об'єкта Ingredients
        }
        // метод для отримання інгредієнтів
        public Ingredients GetIngredients()
        {
            return ingredients;
        }
        // метод для обчислення загальної вартості
        public double CalculateCost()
        {
            return ingredients.CalculateTotalCost();
        }
    }
```

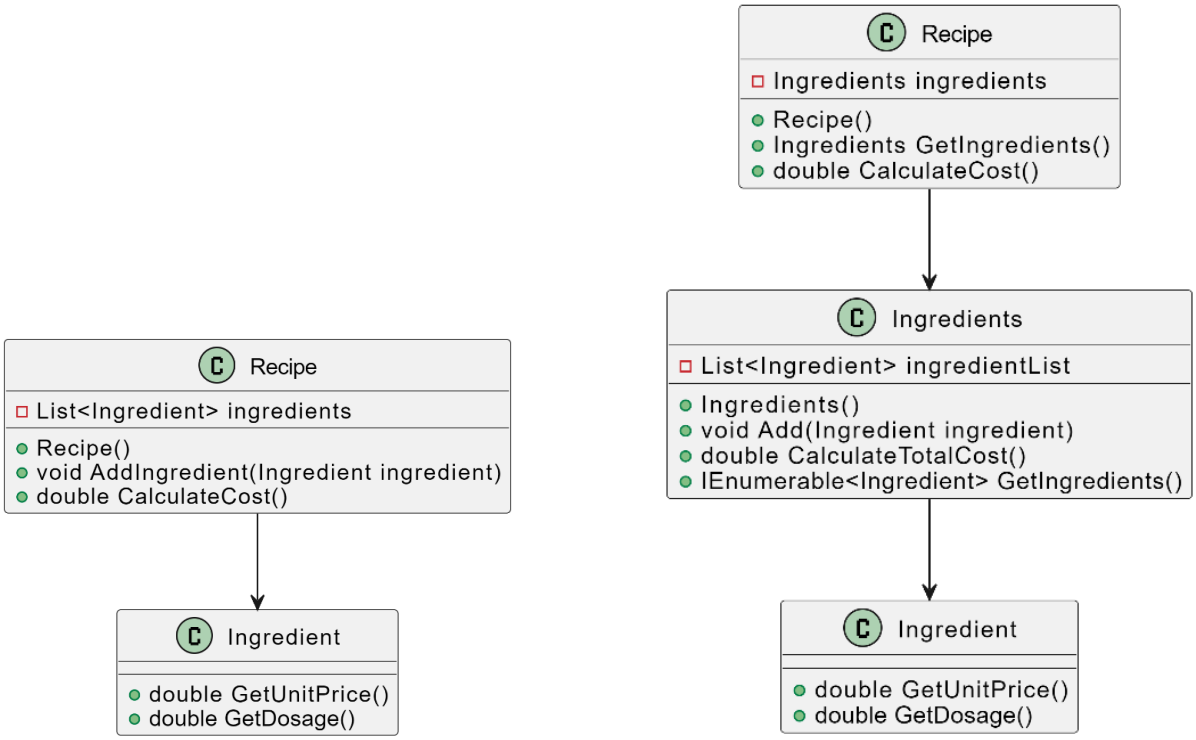

Рисунок 2.12 – Фрагменти UML-діаграм до та після прийому заміна поля-масиву об'єктом

Опис змін в коді після прийому рефакторингу: у брудному коді список інгредієнтів List<Ingredient> знаходився безпосередньо в класі Recipe. У чистому коді створено новий клас Ingredients, що відповідає за управління колекцією інгредієнтів. Це дозволяє розділити відповідальність між класами, спрощуючи їхнє використання. У класі Ingredients реалізовано логіку, пов'язану з обробкою інгредієнтів (додавання, обчислення загальної вартості), що робить код чистішим і зрозумілішим. Клас Recipe тепер зосереджений на своїй основній функції, а управління інгредієнтами віддано окремому класу.

5. Прийом заміна двонаправленого зв'язку одностороннім (Change Bidirectional Association to Unidirectional). Проблема: у вас є двосторонній зв’язок між класами, але один з класів більше не використовує функціонал іншого. Рішення: приберіть невживаний зв’язок.

Фрагмент брудного коду мовою C#:

```
public class Produce
{
    private string name; // приватне поле для назви продукту

    private double unitPrice; // приватне поле для ціни одиниці продукту

    public Produce(string name, double unitPrice)
    {
        this.name=name; // встановлення назви продукту через метод
        this.unitPrice=unitPrice; // встановлення ціни одиниці продукту через метод
    }

}

public class Ingredient : Produce
{
    public double dosage; // поле для дозування
    public Recipe recipe; // двонаправлений зв'язок до рецепту


}

public class Recipe
{
    private List<Ingredient> ingredients; // список інгредієнтів
    public Recipe()
    {
        ingredients = new List<Ingredient>(); // ініціалізація списку інгредієнтів
    }
    public void AddIngredient(Ingredient ingredient)
    {
        ingredients.Add(ingredient);
    }

}
```

Фрагмент чистого коду мовою C#:

```
public class Produce
{
    private string name; // приватне поле для назви продукту
    private double unitPrice; // приватне поле для ціни одиниці продукту

    public Produce(string name, double unitPrice)
    {
        this.name=name; // встановлення назви продукту через метод
        this.unitPrice=unitPrice; // встановлення ціни одиниці продукту через метод
    }
}
public class Ingredient : Produce
{
    private double dosage; // приватне поле для дозування

    public Ingredient(string name, double unitPrice, double dosage)
        : base(name, unitPrice)
    {
        this.dosage = dosage; // встановлення дозування через конструктор
    }
    // метод для отримання дозування

}
public class Recipe
{
    private List<Ingredient> ingredients; // список інгредієнтів
    public Recipe()
    {
        ingredients = new List<Ingredient>(); // ініціалізація списку інгредієнтів
    }
    public void AddIngredient(Ingredient ingredient)
    {
        ingredients.Add(ingredient);
    }
}
```

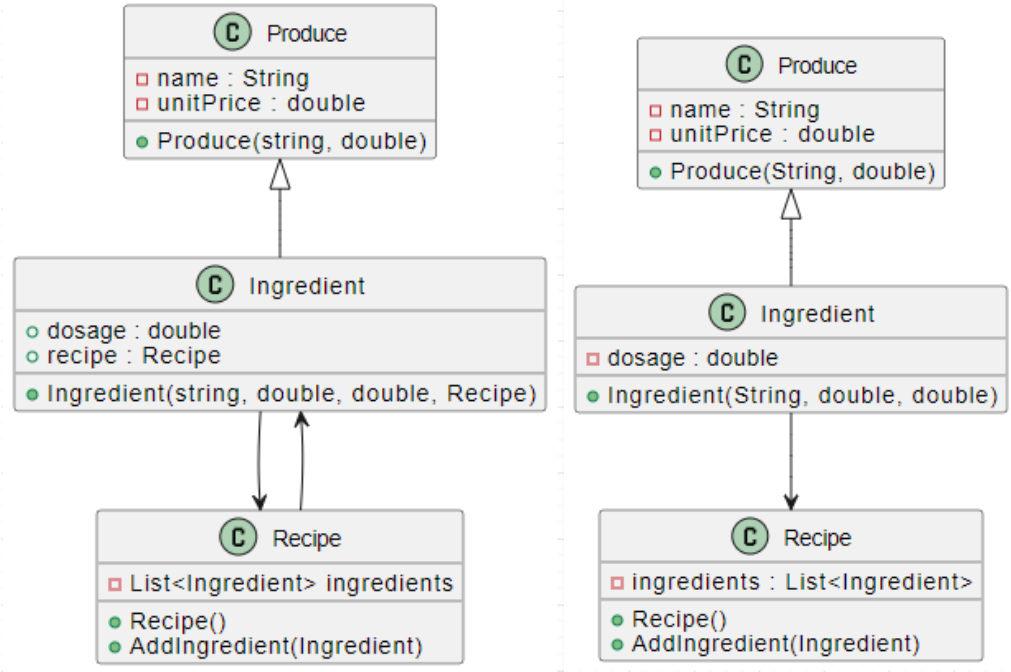

Рисунок 2.13 – Фрагменти UML-діаграм до та після прийому заміна двонаправленого зв'язку одностороннім

Опис змін в коді після прийому рефакторингу: у брудному коді клас Ingredient мав поле Recipe, яке вказувало на відповідний рецепт. Це створювало двонаправлений зв'язок, що ускладнювало взаємодію між класами, оскільки зміни в одному класі могли непередбачувано вплинути на інший. Наприклад, при видаленні інгредієнта із рецепту потрібно було додатково обробляти його видалення з Ingredient, що призводило до збільшення обсягу коду і підвищення ймовірності помилок. У чистому коді двонаправлений зв'язок між класами було замінено на односторонній. Клас Ingredient більше не містить посилання на Recipe. Замість цього, інгредієнти просто додаються до списку в класі Recipe. Це означає, що Recipe володіє інгредієнтами, але інгредієнти не знають про свій зв'язок з рецептом.

Висновки до розділу

У розділі виконано аналіз характеристик об'єктів галузі проведення рефакторингу об’єктно-орієнтованої ієрархії класів прикладних програмних систем. Зокрема, виконано аналіз силабусів навчальних дисциплін, що вивчають рефакторинг коду.

Виконана постановка нових лабораторних робіт:

1. група прийомів рефакторингу “Складання методів”;
2. група прийомів рефакторингу “Переміщення функцій між об'єктами”;
3. група прийомів рефакторингу “Організація даних”.

Наведено приклади виконання лабораторних робіт та оформлення звітів, що містять опис предметної області, UML-діаграми ієрархії класів брудного та чистого кодів, фрагменти брудного та чистого кодів мовою програмування C#, опис змін в коді після прийому рефакторингу.

Опанування здобувачами вищої освіти запропонованого лабораторного практикуму забезпечить отримання програмних результатів навчання, які сприяють широкому діапазону їх професійної діяльності та високій конкурентоспроможності на ринку праці.

РОЗДІЛ 3 ВИМОГИ ДО КАДРОВОГО ЗАБЕЗПЕЧЕННЯ ОБ'ЄКТУ ГАЛУЗІ

У сучасних умовах розробка якісного програмного забезпечення стає вирішальним чинником успіху для компаній. Одним із ключових інструментів, який дозволяє забезпечити високу якість коду, є рефакторинг. Ця практика відіграє важливу роль у методології DevOps, оскільки сприяє зниженню технічного боргу, підвищує продуктивність системи та її здатність до адаптації в умовах швидких змін.

У цьому контексті все більш актуальним стає питання підготовки кадрів, здатних ефективно працювати над рефакторингом коду, а також аналізу компетенцій, необхідних для успішного виконання цієї діяльності.

DevOps – це методологія розробки програмного забезпечення, яка поєднує етапи розробки (Development) і операційного забезпечення (Operations) для забезпечення швидкої доставки якісного програмного продукту. Основна мета DevOps полягає у створенні безперервного процесу інтеграції, тестування та розгортання програмного забезпечення.

Рефакторинг як невід’ємна частина цього процесу передбачає покращення структури коду без зміни його функціональності. Це не тільки сприяє зниженню технічного боргу, але й допомагає забезпечити стабільність і ефективність програмного продукту на всіх етапах його життєвого циклу.

Для ефективного рефакторингу коду фахівцям DevOps необхідно мати широкий спектр технічних знань.

1. Мови програмування. Фахівці повинні добре володіти сучасними мовами програмування, такими як Python, Java, C#, JavaScript, які є основними у розробці DevOps-рішень.

2. Інструменти автоматизації. Знання таких інструментів, як Jenkins, GitLab CI/CD, Ansible, є обов’язковим. Вони дозволяють прискорити процеси автоматизації розгортання програмного забезпечення.

3. Контейнеризація. Робота з Docker і Kubernetes – ключові навички для

створення гнучких і адаптивних середовищ розробки.

4. Інтеграція та моніторинг. Використання інструментів для моніторингу та логування, таких як Prometheus, Grafana або ELK Stack, є важливим для забезпечення стабільності та прогнозованості роботи системи.

Окрім технічних навичок, сучасні фахівці повинні володіти також важливими soft skills, зокрема:

– командна робота: здатність взаємодіяти з розробниками, аналітиками й менеджерами проектів для досягнення спільних цілей;

– гнучкість і адаптивність: швидке реагування на зміну вимог проекту та прийняття рішень в умовах невизначеності;

– вміння вирішувати проблеми: у процесі рефакторингу необхідно не тільки виявляти дефекти коду, але й оперативно знаходити оптимальні рішення для їх усунення.

Підготовка DevOps-фахівців потребує значних інвестицій у формальну освіту та професійний розвиток. Для цього необхідні такі знання.

1. Формальна освіта. Ґрунтовні знання у сфері комп’ютерних наук, програмування та IT-інфраструктури є базовими для роботи у сфері DevOps.

2. Сертифікація. Участь у сертифікаційних програмах, таких як AWS, Google Cloud, Docker, Kubernetes, дозволяє підтримувати високий рівень професійної кваліфікації.

3. Стажування та навчальні програми. Важливо забезпечити практичний досвід, який дозволить майбутнім фахівцям глибше розуміти реальні виклики та знаходити оптимальні рішення.

4. Менторство. Підтримка з боку досвідчених фахівців є критично важливою для успішної інтеграції молодих спеціалістів у професійне середовище.

Незважаючи на високий попит на DevOps-фахівців, ця сфера стикається з рядом викликів. Попит на спеціалістів, які володіють глибокими знаннями у рефакторингу та забезпеченні якості коду, значно перевищує їхню пропозицію.

DevOps-фахівці повинні постійно навчатися та адаптуватися до нових технологій, таких як автоматизація процесів за допомогою штучного інтелекту та машинного навчання.

Від спеціалістів очікують одночасно широкого технічного кругозору й глибокої експертизи у вузькопрофільних галузях.

Серед ключових тенденцій у сфері DevOps можна виділити такі.

1. Автоматизація. Інтеграція технологій штучного інтелекту для автоматизації рутинних процесів, включно з рефакторингом.

2. Зростання попиту. Бізнеси все більше цінують DevOps-фахівців, здатних швидко адаптуватися до змін у технологічному середовищі.

3. Співпраця між освітою та бізнесом. Компанії активно співпрацюють із закладами освіти для розробки навчальних програм, які відповідають реальним потребам ринку.

Для підвищення якості підготовки DevOps-фахівців важливо враховувати такі рекомендації:

– розробка комплексних програм навчання: особливий акцент слід зробити на практичних навичках, які студенти зможуть застосовувати у реальних проектах;

– співпраця з бізнесом: тісна взаємодія між освітніми установами та компаніями дозволить забезпечити актуальність навчальних матеріалів;

– інтеграція сучасних технологій: у навчальні програми необхідно включати курси з використання інструментів DevOps, а також вивчення автоматизованих підходів до рефакторингу;

– підтримка молодих фахівців: важливо створити умови для наставництва та практичного стажування, що сприятиме професійному розвитку випускників.

Для забезпечення якісної підготовки фахівців у сфері DevOps освітні програми мають бути практикоорієнтованими, включати реальні кейси та проекти, які допоможуть студентам розвинути прикладні навички. Важливим є партнерство з бізнесом, адже співпраця з IT-компаніями сприятиме розробці

актуальних навчальних матеріалів і методик. Не менш значущою є інтеграція новітніх технологій, оскільки студенти повинні вивчати сучасні інструменти, що широко використовуються на ринку. Крім того, підтримка менторства шляхом залучення досвідчених спеціалістів до навчального процесу допоможе молодим фахівцям ефективніше інтегруватися в професійне середовище.

Розвиток DevOps супроводжується новими трендами, які впливають на галузь. По-перше, це використання штучного інтелекту та машинного навчання дозволяє оптимізувати рефакторинг і тестування коду. По-друге, бізнеси продовжують збільшувати інвестиції у DevOps для покращення ефективності своєї діяльності. Використання інструментів для управління хмарними середовищами, таких як Terraform, стає все більш поширеним.

Розвиток методології DevOps та впровадження рефакторингу як основного інструменту для підтримки якісного коду є визначальними факторами успіху у сучасному програмному забезпеченні. Підготовка компетентних фахівців у цій сфері вимагає інвестицій у освіту, сертифікацію та безперервне навчання. Співпраця бізнесу та освітніх установ, інтеграція сучасних технологій і підтримка молодих спеціалістів є ключовими умовами для досягнення цієї мети.

Кадрове забезпечення DevOps є перспективним напрямом, який дозволяє ефективно адаптуватися до викликів сучасного технологічного світу та відкриває широкі можливості для подальшого розвитку.

Висновки до розділу

Аналіз ключових аспектів кадрового забезпечення ринку DevOps та рефакторингу показав таке:

– необхідність наявності висококваліфікованих кадрів з технічними і софт-скілами, які відповідають вимогам DevOps та рефакторингу;

– важливість постійного розвитку та навчання фахівців для підвищення їхньої конкурентоспроможності на ринку праці.

РОЗДІЛ 4 МЕТОДИКА ПРОФЕСІЙНОЇ ПІДГОТОВКИ ФАХІВЦІВ З ЦИФРОВИХ ТЕХНОЛОГІЙ ДЛЯ ВИКЛАДАННЯ ОСВІТНЬОГО МОДУЛЯ «РЕФАКТОРИНГ ПРОГРАМНОГО КОДУ» У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ». ДИДАКТИЧНИЙ ПРОЕКТ КОНСУЛЬТАТИВНОГО ЗАНЯТТЯ З ТЕМИ «ВИКОНАННЯ ПРИЙОМІВ РЕФАКТОРИНГУ У VISUAL STUDIO» ДИСЦИПЛІНИ «АНАЛІЗ ТА РЕФАКТОРИНГ КОДУ» ДЛЯ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ СПЕЦІАЛЬНОСТІ 015 ПРОФЕСІЙНА ОСВІТА (ЦИФРОВІ ТЕХНОЛОГІЇ)

4.1 Вихідні дані

Навчальний заклад: Бахмутський навчально-науковий професійно-педагогічний інститут Харківського національного університету імені В.Н. Каразіна;

галузь знань: 01 Освіта / Педагогіка;

спеціальність: 015 Професійна освіта (Цифрові технології);

рівень вищої освіти: перший (бакалаврський);

освітній ступінь: бакалавр;

дисципліна: «Аналіз та рефакторинг коду»;

тема: «Виконання прийомів рефакторингу у Visual studio».

Дисципліна містить такі характеристики, як:

кількість кредитів – 6.

Загальна кількість годин для вивчення дисципліни – 120 навчальних години з яких 78 годин самостійної роботи та 42 години аудиторної (12 годин лекційних занять, 30 годин лабораторної роботи) для денної форми навчання.

Загальна кількість годин для вивчення дисципліни – 120 навчальних години з яких 108 годин самостійної роботи та 12 годи аудиторної (4 години лекційних занять та 8 годин лабораторної роботи) для заочної форми навчання.

Співвідношення кількості годин аудиторних занять до самостійної і індивідуальної роботи становить:

– для денної та заочної форми навчання – 42/78;

– для заочної форми навчання – 12/108.

Дисципліна «Аналіз та рефакторинг коду» викладається у 5-му семестрі 3-го року професійної підготовки бакалаврів для денної та заочної форм навчання та 7-му 4-го року професійної підготовки бакалаврів для заочної форм навчання.

Форми контролю: екзамен.

Великий обсяг навчального матеріалу, обширні, складні цілі навчання та великий відсоток часу, що відведено на самостійну роботу, обумовлюють необхідність у проведенні консультативних занять для уточнення та пояснення навчального матеріалу з дисципліни «Аналіз та рефакторинг коду».

4.2 Проектування цілей консультативного заняття

Визначення навчальних цілей заняття є одним з головних питань, від вирішення якого залежить методична організація заняття, вибір його форм, методів, засобів навчання та контролю. В цьому сенсі навчальні цілі є системоутворюючим фактором, бо, відображуючи кінцеві необхідні результати досягнень майбутніх фахівців, чітко детермінують принципи побудови системи навчання по всіх основних її параметрах.

Тому методична підготовка майбутніх фахівців передбачає перш за все оволодіння вміннями диференційовано визначати навчальні цілі, відповідно до певних рівнів професійної підготовки або рівнів засвоєння.

Проектування цілей консультативного заняття представлені у табл. 4.1 [54]. Таким чином, нами були розроблені цілі консультативного заняття з теми «Виконання прийомів рефакторингу у Visual studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології).

Таблиця 4.1

Цілі консультативного заняття

Цілі консультативного заняття	Цілі формування різних рівнів засвоєння навчального матеріалу	Умови досягнення	Результат у вигляді дій здобувачів вищої освіти
1	З переліку визначень впізнавати основні поняття теми «Виконання прийомів рефакторингу у Visual Studio» такі, як прийоми рефакторингу групи «Складання методів», прийоми рефакторингу групи «Переміщення функцій між об'єктами», прийоми рефакторингу групи «Організація даних», вміти називати зміни які здійснює рефакторинг.	Знати визначення понять «Інкапсуляція поля», «Виділення інтерфейсу», «Виділення класу», «Виділення підкласу», «Виділення надкласу», «Виділення локальної змінної» та знання сутності поняття «Рефакторинг».	Правильно названі з переліку основні поняття теми «Виконання прийомів рефакторингу у Visual Studio» такі, як прийоми рефакторингу групи «Складання методів», прийоми рефакторингу групи «Переміщення функцій між об'єктами», прийоми рефакторингу групи «Організація даних», вміти називати зміни які здійснює рефакторинг.
2	Уміти розповідати про основні принципи виконання прийомів рефакторингу у Visual Studio; характеризувати функціональні можливості рефакторінгу; описувати методи, класи, підкласи, інтерфейси та прийоми рефакторингу.	Виконання дій першого рівня: правильно названі з переліку основні поняття теми ««Виконання прийомів рефакторингу у Visual Studio» такі, як прийоми рефакторингу групи «Складання методів», прийоми рефакторингу групи «Переміщення функцій між об'єктами», групи «Організація даних», вміти називати зміни які здійснює рефакторинг.	Наведена правильна і повна розповідь про основні принципи виконання прийомів рефакторингу у Visual Studio; характеризувати функціональні можливості рефакторінгу; описувати методи, класи, підкласи, інтерфейси та прийоми рефакторингу.
3	Уміти аналізувати переваги й недоліки рефакторінгу у Visual Studio. Уміти характеризувати його виділення.	Виконання дій першого і другого рівнів.	Правильно проаналізовано переваги й недоліки прийомів рефакторингу у Visual Studio. Правильно дана характеристика прийомів рефакторингу.
4	Уміти розробляти та виконувати прийоми рефакторингу у Visual Studio.	Виконання дій першого, другого і третього рівнів.	Правильно розроблено та виконуано прийоми рефакторингу у Visual Studio.

4.3 Перелік джерел інформації

Майбутній фахівець має вміти самостійно користуватися джерелами інформації.

Наведемо перелік джерел інформації для підготовки здобувачів вищої освіти до консультації згідно з робочою програмою дисципліни «Аналіз та рефакторинг коду».

Нижче представлено перелік основної та допоміжної літератури, а також інформаційні ресурси для вивчення дисципліни та підготовки до консультативного заняття.

Рекомендована література:

Основна (базова) література

1. Ed Crookshanks. Just Enough Debugging: Debugging, Refactoring, and Unit Tests: book. CreateSpace Independent Publishing Platform. 2021.

2. Rival X., Kwangkeun Yi. Introduction to static analysis: an abstract interpretation perspective. Mit Press, 2020.

3. Бородкіна І. Інженерія програмного забезпечення: навчальний посібник. Центр учбової літератури, 2021. 204 с.

4. Роберт Мартін. Чистий код. Створення і рефакторинг за допомогою Agile. Фабула, 2019. 368 с. ISBN 978-617-09-5285-1

5. Швець О. Занурення в патерни проєктування : підручник. К.: Refactoring.Guru, 2021. 393 с. URL: https://refactoring.guru/files/design-patterns-ru-demo.pdf.

Додаткова (допоміжна) література

1. Arooj A., Zeeshan A. Refactoring of Code to Remove Technical Debt and Reduce Maintenance Effort. 14th International Conference on Open Source Systems and Technologies (ICOSST). 2020.

2. Cruz I., Mirolo C. Asking Students to Refactor their Code: A Simple and Valuable Exercise. Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. 2024. P. 73-79.

3. Prokop Y., Trofymenko O., Zadereyko A. Developing code style skills in students. IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), October 19-21, 2023, Lviv, Ukraine.

4. Řechtáčková A., Pelánek R., Effenberger T. Catalog of Code Quality Defects in Introductory Programming. Proceedings of the 2024 on Innovation and Technology in Computer Science Education. 2024. V. 1. P. 59-65.

5. Ганчук Н.А. Розробка програмного засобу для ефективного рефакторингу та контрою якості архітектури проєкту. BS thesis. Тернопільський національний технічний університет імені Івана Пулюя, 2024.

6. Зіарманд Д. Н., А. О. Гаврашенко Розробка веб-системи автоматичного аналізу та виправлення програмного коду з використанням штучного інтелекту. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матер. 14 міжнар. наук.-техн. конф., 25-26 квітня 2024 р. Харків : Impress., 2024. С. 19.

7. Колиняк В. М. Аналіз та розробка ефективних методів DevOps для оптимізації процесу розробки програмного забезпечення : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 Інженерія програмного забезпечення. Тернопіль, 2024. 73 с.

8. Струзік В. А. Вдосконалення технологій проведення рефакторингу баз даних для інформаційних систем: автореф. дис. ... канд. техн. наук : 05.13.06 "Інформаційні технології"; Нац. ун-т харч. технол. Київ, 2020. 25 с.

9. Ткачук А. В. Автоматизація рефакторингу коду із використанням бази знань і логічних правил. Науковий вісник НЛТУ України 34.2, 2024. С 87-93.

10. Чикунов П.О. Відстеження поширення по програмі неперевірених зовнішніх даних / Європейські орієнтири розвитку України: науково-практичний вимір в умовах воєнних викликів : матеріали Міжнар.наук.-практ. конф. (Одеса, 26 квітня 2024 р.). Одеса : Фенікс, 2024. С. 866-869.

11. Чикунов П.О. Застосування патернів проектування у процесі конструювання програмного забезпечення. Європейські орієнтири розвитку України в умовах війни та глобальних викликів XXI століття: синергія

наукових, освітніх та технологічних рішень : у 2 т. : матер. Міжнар. наук.-практ. конф. (19 травня 2023 р.) Одеса : Видавництво «Юридика», 2023. Т. 1. С. 606-608.

12. Чикунов П.О. Рефакторинг коду при конструюванні програмного забезпечення. Актуальні тенденції розвитку освіти, науки та технологій : матер. VI Міжнар. наук.-практ. конф. (18 травня 2023 р.). : у 2-х ч. Бахмут – Харків: ННППІ УІПА, 2023. Ч 1. С. 98-100.

Інформаційні ресурси

1. http://cyber.onua.edu.ua/ – робочі матеріали з курсу
2. http://dspace.onua.edu.ua – eNUOLAIR – депозитарій (архів) НУ «ОЮА»
3. Занурення в рефакторинг. Онлайн-курс про поліпшення коду за допомогою рефакторинга. URL: https://refactoring.guru/uk/refactoring/course
4. Agile Methodology & Model: Guide for Software Development & Testing. URL: https://www.guru99.com/agile-scrum-extreme-testing.html
5. Roslyn: майстерність статичного аналізу. URL: https://www.pluralsight.com/courses/refactoring-fundamentals
6. Чому рефакторинг – це постійний процес. URL: https://dou.ua/lenta/columns/why-refactoring-is-important/
7. Що таке рефакторинг коду і чому він важливий для тестувальників. URL: https://training.qatestlab.com/blog/technical-articles/what-is-code-refactoring-and-why-is-it-important-for-testers/
8. 10 Tips for Writing Clean Code. URL: https://www.pluralsight.com/blog/software-development/10-steps-to-clean-code
9. The top three clean code principles to follow in 2023. URL: https://levelup.gitconnected.com/the-top-three-clean-code-principles-to-follow-in-2023-87d7a36407a
10. The .NET Compiler Platform SDK – MS Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/

11. Software Engineering. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series. http://sites.computer.org/ccse/SE2004Volume.pdf

12. IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. – http://standards.ieee.org/reading/ieee/std_public/ description/se/610.12-1990_desc.html.

13. Glossary of Software Engineering terms. – http://www.shellmethod.com/refs/ seglossary.pdf.

Основним джерелом для підготовки здобувачів вищої освіти до консультації є конспект лекцій, розроблений викладачем, оскільки він є найбільш адаптованим до робочої програми дисципліни «Аналіз та рефакторинг коду».

4.4 Визначення найбільш складних для розуміння та засвоєння питань

На даному етапі визначимо найбільш складні для розуміння та засвоєння питання (табл. 4.2) [54].

Отже, на даному етапі ми визначили найбільш складні для розуміння та засвоєння питання з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології).

4.5 Вибір дидактичних методів активізації

На наступному етапі, оберемо методи активізації навчальної діяльності здобувачів вищої освіти на консультації (табл. 4.3) [55].

Таким чином, ми обрали методи активізації навчальної діяльності на консультації.

Таблиця 4.2

Обрання питань для консультування та формулювання відповідей на можливі питання

Теми (або тема) дисципліни	Зміст програми за кожною темою	Найбільш складні питання за темами (темою)	Відповіді на питання
Виконання прийомів рефакторингу у Visual Studio.	1. Прийоми рефакторингу групи «Складання методів» 2. Прийоми рефакторингу групи «Переміщення функцій між об'єктами» 3. Прийоми рефакторингу групи «Організація даних»	1. В чому полягає зміна сигнатури методу ?	1. Зміна сигнатури методу передбачає додавання, зміну або видалення параметрів методу. У контексті ООП сигнатура визначається назвою методу, а також кількістю, порядком і типами його параметрів, причому тип поверненого значення до сигнатури не входить. При зміні сигнатури необхідно внести відповідні корективи у виклики цього методу в коді всіх залежних клієнтів. Така модифікація може вплинути на зовнішній інтерфейс програми. Якщо доступ до всіх клієнтів інтерфейсу обмежений, може знадобитися використання механізмів для повідомлення про внесені зміни.
		2.Що таке інкапсуляція поля ?	2. Інкапсуляція поля (Encapsulate Field) – це прийом, спрямований на спрощення роботи з даними, підвищення модульності компонентів та полегшення її обслуговування. Він дозволяє об'єднати дані з поведінкою, яка їх обробляє. Якщо в класі існує відкрите поле, його слід зробити закритим і надати методи доступу.
		3. Що дозволяє створювати інкапсуляція поля.	3. Інкапсуляція поля полягає у перетворенні існуючого поля на властивість і поступовому оновленні коду для використання цієї властивості. Якщо в класі присутнє відкрите поле, його слід зробити закритим і надати методи доступу до нього.

Таблиця 4.3

Методи активізації навчальної діяльності здобувачів вищої освіти на консультації

Дидактичні методи	Реалізація методів при проведенні консультаційного заняття
Методи підвищення наочності	Використання інтерактивної дошки та мультимедійного проектора для демонстрації слайдів з теми «Виконання прийомів рефакторингу у Visual Studio», та використання плакатів для демонстрації прийомів рефакторингу групи «Складання методів», «Переміщення функцій між об'єктами» та «Організація даних».
Мотиваційні методи	Для реалізації мотивації використаємо: тип: внутрішня мотивація; вид: вступна мотивація; метод: мотивуючий вступ; прийом: віднесення до особистості. Повідомлення важливості вивчення даної теми: «Виконання прийомів рефакторингу у Visual Studio» є дуже важливою, оскільки на підприємствах існує безліч ситуацій, в яких програміст має виявити власну компетентність. Тому для вас, як майбутніх програмістів, ця тема є достатньо актуальною. Відповідно до вашої майбутньої професійної діяльності знання цього навчального матеріалу знадобляться вам для програмування на підприємстві, а саме це дозволить ефективно використовувати засоби сучасних інформаційних систем. Створення баз даних за допомогою та їх застосування для розв'язання певного типу задач є одним з основних завдань вашої майбутньої діяльності. Від того, наскільки успішно ви справитеся з даним завданням, вже працюючи на підприємстві, залежить його благополуччя, а вам дозволить рости у якості високоякісного програміста в області програмування та стверджуватимуть вас як висококваліфікованого фахівця».
Проблемні методи	Використання проблемного питання. Проблемне питання: «Для чого використовуються операції «Перейменування»?
Комунікативні методи	Основною рисою комунікативного методу являється комунікативна складова, яка включає в себе цілий ряд характеристик: вступ, що зацікавлює – важливий елемент мовлення, який повинен враховувати інтерес аудиторії та налаштовувати на подальше прослуховування; чітка дикція – необхідна для того, щоб слухачі швидко і легко сприймали інформацію; уміння володіти собою – допомагає слухачам зосередитися на темі доповіді, а не на виступаючому; зоровий контакт – розглядається як показник впевненості в своїх словах та зацікавленості в слухачах; сила голосу – враховує певні особливості, а саме: 1) кількість слухачів, 2) зовнішні шуми, 3) зміст доповіді, 4) мета доповіді; розкриття теми – надає доповіді цілісності, допомагає слухачам зрозуміти та краще запам'ятати те, про що говориться; переконлива мова – допомагає слухачам віднестися уважно до матеріалу доповіді.

4.6 Вибір способів організації консультативного заняття

Далі необхідно здійснити вибір способів організації консультативного заняття. Він здійснюється з урахуванням даних, наведених в таблиці 4.4 [53].

Таблиця 4.4

Варіанти організації консультативного заняття

№ варіанта	Етапи організації заняття	Характеристика варіанта
1	2	3
1	- вступне слово лектора, - відповіді на питання здобувачів вищої освіти і обговорення їх, - заключне слово викладача	Недоліком цього варіанту проведення лекції-консультації є відсутність послідовності, системи в питаннях, на які доводиться викладачу давати відповіді. Питання поступають хаотично, що знижує якість консультації.
2	- збір питань в письмовій формі до лекції, їх систематизація, - відповіді на питання, що поступили, - відповіді на додаткові питання, - обмін думками, - висновки	Цей варіант, на відміну від попереднього, дозволяє викладачу групувати відповіді, що сприяє кращому засвоєнню навчального матеріалу здобувачами вищої освіти.
3	- видача завдань на самостійне вивчення матеріалу теми. - підготовка питань лектору. - відповіді і їх обговорення	В цьому випадку консультування грає функцію додаткового інформування зі складних питань і пояснення незрозумілого навчального матеріалу.
4	- повідомлення теми, - консультування декількома фахівцями в певній області науки і техніка з актуальних питань науки і нової техніки	Цей варіант лекції-консультації проводиться, як правило, зі спеціальних дисциплін, іноді для цієї мети використовуються наукові семінари. Такі заняття дають можливість зіставити думки різних учених на одну і ту ж проблему і є чудовою школою ведення дискусії.

Згідно представленої таблиці обираємо 1 варіант організації консультативного заняття, на якому викладач пояснює питання, які здалися незрозумілими здобувачам вищої освіти.

4.7 Розробка сценарію проведення консультативного заняття

На наступному етапі наводимо розробку сценарію проведення консультативного заняття у відповідності до обраного варіанту його організації (табл. 4.5) [55].

Таблиця 4.5

Сценарій консультативного заняття

Етапи проведення консультативного заняття	Дії викладача	Дії здобувачів вищої освіти
1	2	3
Організаційний момент	Викладач вітає здобувачів вищої освіти, робить перекличку, пропонує здобувачам вищої освіти розпочати роботу на консультації.	Здобувачі вищої освіти вітають викладача, беруть участь у перекличці, налаштовуються на роботу на консультації.
Повідомлення теми і мети уроку	Повідомлення теми заняття «Виконання прийомів рефакторингу у Visual Studio», формулювання його цілей: Сформувати знання про основні прийоми рефакторингу у Visual Studio.	Фіксація теми, сприйняття цілей, представлення результатів засвоєння матеріалу теми даного заняття
Мотивація мети	Повідомлення про важливість вивчення теми: «Виконання прийомів рефакторингу у Visual Studio» є надзвичайно актуальним, оскільки у професійній діяльності програміста часто виникають ситуації, які вимагають демонстрації високого рівня компетентності. Для вас, як майбутніх програмістів, ця тема є важливою, адже вона безпосередньо стосується вашої майбутньої професії. Знання цього матеріалу стане у пригоді під час роботи на підприємствах, дозволяючи ефективно застосовувати сучасні інструменти інформаційних систем. Рефакторинг програмного коду та використання прийомів рефакторингу у Visual Studio є одними з ключових завдань, які ви виконуватимете у своїй роботі. Від того, наскільки успішно ви опануєте ці навички, залежить ефективність роботи підприємства.	Сприйняття важливості і актуальності вивчення теми, прояв інтересу до неї.

Продовження табл. 4.5

1	2	3
Актуалізація знань	Викладач проводить фронтальне усне опитування з метою перевірки базових знань:	Здобувачі вищої освіти беруть участь в опитуванні та надають відповіді на поставлені запитання. Передбачені варіанти відповідей:
	1. Як ви розумієте термін «рефакторинг»?	1. Рефакторинг – це процес переписування програмного коду з метою покращення його внутрішньої структури та стилю, без зміни зовнішньої поведінки програми. Він включає зміни, пов’язані з методами, переміщенням функцій між об’єктами, організацією даних, спрощенням умовних виразів і викликів методів, а також вирішення задач узагальнення (наприклад, виділення інтерфейсу, підкласу чи батьківського класу) [57].
	2.Поясніть в чому полягає зміна сигнатури методу?	2. Зміна сигнатури методу передбачає додавання, зміну або видалення параметрів методу. У об’єктно-орієнтованому програмуванні сигнатура визначається назвою методу, кількістю, порядком і типами його параметрів, але не включає тип поверненого значення. Змінивши сигнатуру методу, необхідно оновити всі виклики цього методу у коді клієнтів. Така зміна може вплинути на зовнішній інтерфейс програми, і якщо програмісту недоступні всі клієнти інтерфейсу, можуть знадобитися механізми для реєстрації змін [53].
	3. В чому полягає виділення методу?	3. Виділення методу полягає у відокремленні окремих фрагментів із громіздкого або складного для розуміння коду та перетворенні їх на окремі методи (функції). Новий метод містить виділений код, тоді як у початковому коді цей фрагмент замінюється викликом нового методу.

Продовження табл. 4.5

1	2	3
Формування ООД	Викладач проводить консультацію згідно плану, за допомогою методу - пояснення: 1. Прийоми рефакторингу групи «Складання методів». 2. Прийоми рефакторингу групи «Переміщення функцій між об'єктами». 3. Прийоми рефакторингу групи «Організація даних»	Слухають пояснення, конспектують.
Визначення проблемних моментів під час вивчення питань теми та формування ВД	Викладач запитує здобувачів вищої освіти про недоречності, які виникли у них під час самостійного вивчення теми. Викладач відповідає на поставлені запитання. 1. Інкапсуляція поля (Encapsulate Field) – прийом рефакторингу, що дозволяє полегшити роботу з даними, покращити модульність частин програми та полегшити її підтримку. Забезпечує зближення даних і поведінки над цими даними. Якщо в класі є відкрите поле, то необхідно зробити його закритим і забезпечити методи доступу.	Здобувачі вищої освіти запитують: 1. Інкапсуляція поля?
Підведення підсумків	Викладач підводить підсумки проведення консультації: «Сьогодні ми розглянули незрозумілі вам питання теми. Зараз перевіримо як ви засвоїли незрозумілий вам матеріал. Скажіть, що ж таке «Рефакторинг та які зміни він здійснює»?	Здобувачі вищої освіти слухають, відповідають: «рефакторинг – це процес переписування програмного коду з метою поліпшення його внутрішньої структури та стилю. При цьому зовнішня поведінка програми не змінюється. Рефакторинг здійснює зміни, пов’язані з методами, переміщеннями функцій між об’єктами, організацією даних, спрощенням умовних виразів і викликів методів, задачами узагальнення (виділення інтерфейсу, підкласу, батьківського класу тощо).». Здобувачі вищої освіти прощаються.

Отже, на цьому етапі ми розробили сценарій проведення консультативного заняття у відповідності до обраного варіанту його організації.

На заключному етапі представлено контурний конспект з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології).

Конспект – сукупність взаємозв'язаних опорних сигналів теми.

За обсягом інформації, що представляється, конспекти діляться на повні і контурні (опорні), а за способом подання інформації — на плани-конспекти і конспекти-схеми. План-конспект стисло представляє зміст кожного з пунктів плану. Конспект-схема — це ієрархія понять теми, впорядкованих згідно плану і доповнених основними відомостями.

У повному конспекті міститься, переважно, вся нова основна інформація. У контурному (опорному) ж конспекті містяться тільки ключові положення нової основної інформації, виражені за допомогою таблиць, графіків, абревіатури, різного роду позначень, акцентів.

Контурний конспект заняття з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології) представлено у Додатку.

Висновки до розділу

У четвертому розділі розроблено дидактичний проект консультативного заняття з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології).

Сформульовано цілі консультативного заняття. Обрано методи активізації навчальної діяльності здобувачів освіти на консультації. Здійснено вибір способів організації консультативного заняття.

Розроблено сценарій проведення консультативного заняття у відповідності до обраного варіанту його організації.

Проаналізовано джерела інформації для підготовки здобувачів освіти до консультації згідно з робочою програмою дисципліни «Аналіз та рефакторинг коду». Подано список використаних джерел та відповідні посилання.

ВИСНОВКИ

У кваліфікаційній роботі теоретично обґрунтована та перевірена методика професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти та виділено пріоритетні напрямки удосконалення професійних компетентностей.

З’ясовано, що професійна підготовка фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду» у закладах вищої освіти є актуальною у професійній педагогіці.

Проаналізовано ступінь актуальності проблеми професійної підготовки фахівців з цифрових технологій для викладання освітнього модулю «Рефакторинг програмного коду».

У розділі виконано аналіз характеристик об'єктів галузі проведення рефакторингу об’єктно-орієнтованої ієрархії класів прикладних програмних систем. Зокрема, виконано аналіз силабусів навчальних дисциплін, що вивчають рефакторинг коду.

Виконана постановка нових лабораторних робіт:

1. група прийомів рефакторингу “Складання методів”;

2. група прийомів рефакторингу “Переміщення функцій між об'єктами”;

3. група прийомів рефакторингу “ Організація даних”.

Наведено приклади виконання лабораторних робіт та оформлення звітів, що містять опис предметної області, UML-діаграми ієрархії класів брудного та чистого кодів, фрагменти брудного та чистого кодів мовою програмування C#, опис змін в коді після прийому рефакторингу;

Опанування здобувачами вищої освіти запропонованого лабораторного практикуму забезпечить отримання програмних результатів навчання, які сприяють широкому діапазону їх професійної діяльності та високій конкурентоспроможності на ринку праці.

Аналіз ключових аспектів кадрового забезпечення ринку DevOps та рефакторингу показав таке:

– необхідність наявності висококваліфікованих кадрів з технічними і софт-скілами, які відповідають вимогам DevOps та рефакторингу;

– важливість постійного розвитку та навчання фахівців для підвищення їхньої конкурентоспроможності на ринку праці.

Розроблено дидактичний проект консультативного заняття з теми «Виконання прийомів рефакторингу у Visual Studio» дисципліни «Аналіз та рефакторинг коду» для здобувачів вищої освіти спеціальності 015 Професійна освіта (Цифрові технології).

Сформульовано цілі консультативного заняття. Обрано методи активізації навчальної діяльності здобувачів освіти на консультації. Здійснено вибір способів організації консультативного заняття.

Розроблено сценарій проведення консультативного заняття у відповідності до обраного варіанту його організації.

Проаналізовано джерела інформації для підготовки здобувачів освіти до консультації згідно з робочою програмою дисципліни «Аналіз та рефакторинг коду». Подано список використаних джерел та відповідні посилання.

Апробація результатів дослідження виконана під час виконання лабораторного практикуму з дисципліни «Аналіз та рефакторинг коду» бакалаврами спеціальності 122 «Комп’ютерні науки» Національного університету «Одеська юридична академія».

За основними результатами дослідження виконана публікація тез доповідей на конференції:

– Панчук І.М. Вимоги до кадрового забезпечення ринку праці у галузі DevOps // Матеріали VIII міжнародної науково-практичної конференції здобувачів вищої освіти та молодих учених «Студенти та молодь – для майбутнього країни» (м. Харків, 14-15 листопада 2024 р.) [упоряд. Г. Г. Михальченко]. Харків, ХНУ ім. Каразіна, 2024. – С. 87.

СПИСОК ВИКОРИСТОВАНИХ ДЖЕРЕЛ

1. Силабус навчальної дисципліни «Аналіз та рефакторинг коду» освітньо-професійної програми «Інженерія програмного забезпечення» Національного університету «Одеська юридична академія». URL: https://docs.google.com/document/d/1VPuKgo6HvtExgNCpP6-MvrktFTzus7yV/edit?usp=drive_link&ouid=117130216069796876479&rtpof=true&sd=true

2. Силабус навчальної дисципліни «Технології безперервного розроблення та інтеграції програмного забезпечення» освітньої програми «Інтегровані інформаційні системи» Національного технічного університету «Київський політехнічний інститут імені Ігоря Сікорського». URL: https://osvita.kpi.ua/126_OPPB_IIS

3. Силабус навчальної дисципліни «Рефакторинг програмного забезпечення» ОПП «Комп'ютерні науки» Харківського національного університету радіоелектроніки. URL: https://ics.nure.ua/silabus-ukr/Контент Силлабусуосвітньої програми - ІУС.doc

4. Силабус навчальної дисципліни «Паттерни проєктування» ОПП «Інтелектуальний аналіз даних в комп'ютерних інформаційних системах» Чернівецького національного університету ім. Ю. Федьковича. URL: https://kkn.chnu.edu.ua/wp-content/uploads/2024/02/OP_bak_2023_v6_2_final_signed_dates.pdf

5. 10 Tips for Writing Clean Code. URL: https://www.pluralsight.com/blog/software-development/10-steps-to-clean-code

6. Anagnostopoulos A. Hands-On Software Engineering with Golang: Move beyond basic programming to design and build reliable software with clean code. Packt Publishing Ltd, 2020.

7. Arooj A., Zeeshan A. Refactoring of Code to Remove Technical Debt and Reduce Maintenance Effort. 14th International Conference on Open Source Systems and Technologies (ICOSST). 2020.

8. Bass L., Clements P., Kazman R. Software Architecture in Practice. Software Architecture in Practice: Software Architect Practice. Addison-Wesley, 2012.

9. Breivold H., Crnkovic I., Larsson M. A systematic review of software architecture evolution research. Information and Software Technology, 2012, 54.1: Pp. 16-40.

10. Cruz I., Mirolo C. Asking Students to Refactor their Code: A Simple and Valuable Exercise. Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1. 2024. P. 73-79.

11. Ed Crookshanks. Just Enough Debugging: Debugging, Refactoring, and Unit Tests: book. CreateSpace Independent Publishing Platform. 2021.

12. IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. – http://standards.ieee.org/reading/ieee/std_public/ description/se/610.12-1990_desc.html.

13. Most Common Software Architecture Styles. URL: https://medium.com/@techworldwithmilan/most-common-software-architecture-styles-86881d779683

14. Prokop Y., Trofymenko O., Zadereyko A. Developing code style skills in students. IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), October 19-21, 2023, Lviv, Ukraine.

15. Řechtáčková A., Pelánek R., Effenberger T. Catalog of Code Quality Defects in Introductory Programming. Proceedings of the 2024 on Innovation and Technology in Computer Science Education. 2024. V. 1. P. 59-65.

16. Reddit. r/softwarearchitecture. URL: https://www.reddit.com /r/ softwarearchitecture/?rdt=41013

17. Rival X., Kwangkeun Yi. Introduction to static analysis: an abstract interpretation perspective. Mit Press, 2020.

18. Software Engineering. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series. http://sites.computer.org/ccse/SE2004Volume.pdf

19. The .NET Compiler Platform SDK – MS Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/

20. The top three clean code principles to follow in 2023. URL: https://levelup.gitconnected.com/the-top-three-clean-code-principles-to-follow-in-2023-87d7a36407a

21. Бородкіна І. Інженерія програмного забезпечення: навчальний посібник. Центр учбової літератури, 2021. 204 с.

22. Вікіпедія: DevOps . URL: https://uk.wikipedia.org/wiki/DevOps

23. Ганчук Н.А. Розробка програмного засобу для ефективного рефакторингу та контрою якості архітектури проєкту. BS thesis. Тернопільський національний технічний університет імені Івана Пулюя, 2024.

24. Зіарманд Д. Н., А. О. Гаврашенко Розробка веб-системи автоматичного аналізу та виправлення програмного коду з використанням штучного інтелекту. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матер. 14 міжнар. наук.-техн. конф., 25-26 квітня 2024 р. Харків : Impress., 2024. С. 19.

25. Ключові методології розробки програмного забезпечення: робота команди зсередини. URL: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija

26. Колиняк В. М. Аналіз та розробка ефективних методів DevOps для оптимізації процесу розробки програмного забезпечення : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 Інженерія програмного забезпечення. Тернопіль, 2024. 73 с.

27. Оберванюк Н.-П. Б. Методика забезпечення якості при проектуванні архітектури програмного забезпечення в Agile-проектах. Master's Thesis 2020.

28. Поліщук А.М., Ніколюк П.К. Технологія програмування та основні етапи її розвитку. Комп’ютерні технології обробки даних, 2022, 89-91.

29. Роберт Мартін. Чистий код. Створення і рефакторинг за допомогою Agile. Фабула, 2019. 368 с. ISBN 978-617-09-5285-1

30. Самчинська Я.Б., Вінник М.О. Просування й розповсюдження педагогічного програмного забезпечення на ринку України. Інформаційні технології в освіті, 2013. № 15. – С. 210-220.

31. Струзік В. А. Вдосконалення технологій проведення рефакторингу баз даних для інформаційних систем: автореф. дис. ... канд. техн. наук : 05.13.06 "Інформаційні технології"; Нац. ун-т харч. технол. Київ, 2020. 25 с.

32. Струк А. Формування здатності майбутніх інженерів-програмістів до проектування програмного забезпечення. Journal of Information Technologies in Education (ITE), 2022, №50. Pp. 61-79.

33. Ткачук А. В. Автоматизація рефакторингу коду із використанням бази знань і логічних правил. Науковий вісник НЛТУ України 34.2, 2024. С 87-93.

34. Чикунов П.О. Рефакторинг коду при конструюванні програмного забезпечення. Актуальні тенденції розвитку освіти, науки та технологій : матер. VI Міжнар. наук.-практ. конф. (18 травня 2023 р.). : у 2-х ч. Бахмут – Харків: ННППІ УІПА, 2023. Ч 1. С. 98-100.

35. Швець О. Занурення в патерни проєктування : підручник. К.: Refactoring.Guru, 2021. 393 с. URL: https://refactoring.guru/files/design-patterns-ru-demo.pdf.

36. Що таке рефакторинг коду і чому він важливий для тестувальників. URL: https://training.qatestlab.com/blog/technical-articles/what-is-code-refactoring-and-why-is-it-important-for-testers/

37. Антонюк Л.Л. Компетентністний підхід у вищій освіті: світовий досвід навч. пос. Київ : КНЕУ, 2016.66 с.

38. Професійна педагогіка : Підручник / Авт. : О. В. Грабовський, Л. В.Коломієць, О. С. Савельєва, А. В.Семенова, В. Ф.Яні; за заг. ред. А. В.Семенової. – Одеса : Бондаренко М. О., 2020. – 575 с.

39. Професійна педагогіка: навч. посібник для вищих навч. закладів/ В. І. Жигірь, О. Чернєга ; за ред. М. В. Вачевського. - Київ: К.: Кондор, 2016. – 336 с. 978-966-351-359-1. Код 233704.

40. Зайченко І. В. Теорія і методика професійного навчання: навч. посібник. 2-е вид., доповн. і переробл. К.: Видавництво Ліра-К, 2016. 580 с.

41. Методика формування пошуково-дослідницьких умінь майбутніх інженерів-педагогів у процесі професійної підготовки: колективна монографія / В.В. Кулешова, В.В. Мальована. – Артемівськ: ННППІ УІПА, 2012. – 264 с. (власний внесок: Р1; Р2 с.86-92; Р3; 9,8 д.а.).

42. Формування професійної компетентності викладачів технічних дисциплін: колективна монографія / В.В.Кулешова, В.В.Мальована, Ю.С.Бобрикова. – Х., 2020. – 206 с. (власний внесок: Р1 с.8-94; Р2 с.95-100; Р3с.146-159; 6,5 д.а.).

43. Кулешова В.В., Мальована В.В. Особливості особистості викладача технічних дисциплін у вищих навчальних закладах/ Проблеми інженерно-педагогічної освіти. Збірник наукових праць. №50-51 Харків: УІПА,– 2016 р., – С.322-329

44. Кулешова В.В., Мальована В.В. Формування професійних методичних умінь у майбутніх інженерів-педагогів економічного профілю / Міжнародний науковий журнал «ІНТЕРНАУКА». №7 (29) Київ:– 2017 р., – С.26-29

45. Кулешова В.В.Формування креативної компетентності майбутніх інженерів у процесі професійної підготовки / Проблеми інженерно-педагогічної освіти. Збірник наукових праць. № 58 Харків: УІПА,– 2018 р., – С.21-26

46. Коваленко А.В. Шляхи забезпечення формування безперервної освіти: школа – ПЗО – Фаховий коледж – Академія. Роль закладів фахової передвищої та 194 професійної освіти в системі безперервної освіти: матеріали VII наук.-практ. конф., м. Одеса, 25 бер.2020 р. Одеса, 2020. С.21-24.

47. Олійник В. В. Відкрита післядипломна педагогічна освіта: нові моделі та форми професійного розвитку / Освіта дорослих у перспективі змін: інновації, технології, прогнози: колективна монографія / За ред.. А. Василюк,

А. Стоговського. – Ніжин: Видавець ПП Лисенко М.М., 2017. – 248 с. С. 93–108.

48. Ортинський В. Л. Педагогіка вищої школи: навч. посіб. / Ортинський В. Л. – Центр учбової літератури, 2017. – 472 с

49. Професійна освіта України на шляху до євроінтеграції (1992–2017) / науков. ред. Н.Г. Ничкало; упорядники: Л.В. Горбань, В.П. Тименко. – К.: ДП «Інформ.-аналіт. агенство», 2018. – 358 с.

50. Сисоєва С.О. Теорія і практика вищої освіти: навч. посібник / С.О. Сисоєва, І.В. Соколова. – К., 2016. – 338 с. – Режим доступу: http://lib.iitta.gov.ua/711948/1/Sysoieva_Socolova_2016_pos..pdf

51. Теорія та методика викладання фахових дисциплін у ЗВО: навчально- методичний посібник / укладач І. В. Казанжи – Миколаїв : СПД Румянцева, 2018. – 154 с.

52. Теорія і практика вищої професійної освіти в Україні : навч. посіб. для магістрантів зі спеціальності 011 «Освітні, педагогічні науки» / [авт.-укл.: Т.О.Дороніна]. – Кривий Ріг : КДПУ, 2018. – 250 с.

53. Методика професійного навчання: метод. вказ. по виконанню курсової роботи для здобувачів освіти освітнього ступеня «бакалавр» денної та заочної форми навч. інженерно-педагогічних спеціальностей / ННППІ Укр. інж.-пед. акад. ; упоряд. : В. В. Кулешова, В.В. Мальована, Ю.С. Бобрикова ; за заг. ред. д-ра пед. наук, проф.В. В. Кулешової. – Бахмут, 2022. – 92 с.

54. Методика професійного навчання : конспект лекцій для здобувачів вищої освіти ОС «бакалавр» денної та заоч. форм здобуття освіти спец. 015 Проф. освіта (за спеціалізаціями). Ч. 1 / О. Е. Коваленко, Н. О. Брюханова, Н. В. Корольова; Укр. інж.-пед. акад., Каф. педагогіки, методики та менеджменту освіти. - Харків: УІПА, 2020. - 200 с.

55. Методика професійного навчання: конспект лекцій для здобувачів вищої освіти ОС «бакалавр» денної та заоч. форм здобуття освіти спец. 015 Проф. освіта (за спеціалізаціями). Ч. 2/ О. Е. Коваленко, Н. О. Брюханова, Н.

В. Корольова; Укр. інж.-пед. акад., Каф. педагогіки, методики та менеджменту освіти. - Харків: УІПА, 2020. - 180 с.

56. Коваленко О. Е., Брюханова Н. О., Корольова Н.В. Методика професійного навчання: дидактичне проектування: Підручник для студентів інженерно-педагогічних спеціальностей. – Харків: УІПА, 2019. – 204 с.

57. Коваленко О. Е., Брюханова Н. О., Корольова Н.В. Методика професійного навчання: основні технології навчання: Підручник для студентів інженерно-педагогічних спеціальностей. – Харків: УІПА, 2019. – 174 с.

58. Методика професійного навчання: дидактичне проектування: конспект лекцій для студ. денної та заоч. форм навч. спец. 015.06 "Проф. освіта. Електроніка, радіотехн. та телекомунікації" освітнього рівня "бакалавр"/ Н. Г. Кошелева; Укр. інж.-пед. акад. (Бахмут), ННППІ. - Бахмут, 2017. - 100 с.

59. Методика професійного навчання: основні технології навчання: конспект лекцій для студ. денної та заоч. форм навч. спец. 015.06 "Проф. освіта. Електроніка, радіотехн. та телекомунікації" освітнього рівня "бакалавр"/ Н. Г. Кошелева; Укр. інж.-пед. акад. (Бахмут), ННППІ. - Бахмут, 2017. - 101 с.

60. Теорія і методика професійного навчання : навч. посібник. – 2-е вид., доповн. і переробл. / І.В.Зайченко. – К.: Видавництво Ліра-К, 2016. – 580 с.

61. Методика професійного навчання: навч. посібник для вищих навч. закладів інж.-пед. спец. для традиційної та дистанційної форм навчання. Ч. 1: Дидактичне проектування/ О. Е. Коваленко, Н.О. Брюханова, Н.В. Корольова; Укр. інж.- пед. академія. - 2-ге вид., перероб. та доп.. - Х.: ФОП Шевченко С. О., 2010. - 264 с.

ДОДАТОК А. КОНТУРНИЙ КОНСПЕКТ НА ТЕМУ «ВСТУП У РЕФАКТОРИНГ ПРОГРАМНОГО КОДУ»

Рефакторинг – це процес переписування програмного коду з метою поліпшення його внутрішньої структури та стилю. При цьому зовнішня поведінка програми не змінюється.

Рефакторинг здійснює зміни, пов’язані з методами, переміщеннями функцій між об’єктами, організацією даних, спрощенням умовних виразів і викликів методів, задачами узагальнення (виділення інтерфейсу, підкласу, батьківського класу тощо).

Зміна сигнатури методу полягає в додаванні, зміні або видаленні параметра методу. Сигнатура методу в ООП задається назвою методу, кількістю, порядком і типами його параметрів, при цьому тип результату не є частиною сигнатури методу. Змінивши сигнатуру методу, необхідно скорегувати звернення до нього в коді всіх клієнтів. Ця зміна може торкнутися зовнішнього інтерфейсу програми. Якщо програмісту не доступні всі клієнти інтерфейсу, то може знадобитися та чи інша форма реєстрації змін інтерфейсу.

Інкапсуляція поля (Encapsulate Field) – прийом рефакторингу, що дозволяє полегшити роботу з даними, покращити модульність частин програми та полегшити її підтримку. Забезпечує зближення даних і поведінки над цими даними. Якщо в класі є відкрите поле, то необхідно зробити його закритим і забезпечити методи доступу.

Виділення інтерфейсу (Extract Interface) – якщо кілька клієнтів використовують одну підмножину інтерфейсу класу або в кількох класах певна частина інтерфейсу спільна, то рефакторинг забезпечує виділення цієї підмножини в окремий інтерфейс.

Виділення класу – іноді клас реалізує функцію, яку слід розподілити між кількома класами. Тоді рефакторинг дозволяє створити новий клас, перемістити відповідні атрибути й методи зі старого класу в новий.

Виділення підкласу – якщо клас виконує різні види робіт, то доцільно

створити підкласи, які йому спадкують і виконують кожен свій вид роботи.

Виділення надкласу – якщо маємо кілька класів зі схожими методами й полями, то можна перемістити їх у надклас (батьківський для всіх даних підкласів).

Виділення локальної змінної – якщо певний вираз обчислюється в тілі методу в кількох місцях, то його значення слід обчислити один раз і запам'ятати в новій локальній змінній. Наприклад, код

```
func(a+b, (a+b) / 2, a+b-c);
```

слід рефакторизувати в:

```
t=a+b;
func(t, t/2, t-c);
```

попередньо оголосивши `t` локальною змінною відповідного типу.

Вбудовування змінної – підстановка єдиний раз присвоєного значення існуючої локальної змінної замість неї самої. Можна використовувати для економії (стекової) пам'яті, що корисно в рекурсивних функціях. Наприклад, код

```
double y=f(x);
if (y<0) {…}
```

слід рефакторизувати в

```
if (f(x)<0) {…}
```

Виділення методу полягає у виділенні з довгого чи важкого для сприйняття коду окремих його фрагментів і перетворенні їх в окремі методи (функції).

Підйом поля (методу) – якщо у двох підкласах є однакове поле чи метод, то їх слід підняти в батьківський клас.

Спуск поля (методу) – якщо в батьківському класі є поля чи методи, що належать лише до деяких його підкласів, то їх потрібно перемістити в ці підкласи.

Вбудовування методу протилежне виділенню методу. Коли тіло методу (функції) так само зрозуміле, як і його (її) назва, тоді виклик методу замінюють

на код. Крім того, такий рефакторинг застосовують при невдалому розбитті на методи – тоді кілька методів об’єднують, а потім знову виділяють окремі методи.

Заміна умовного оператора поліморфізмом – умовний оператор із кількома розгалуженнями замінюється викликом поліморфного методу деякого базового класу, що має підкласи для кожної гілки вихідного оператора. Вибір гілки здійснюється неявно.

Переміщення методу застосовується для методу, що частіше звертається до іншого класу, ніж до того, у якому він розташований.

```
class Library
{
    public Book[] books = new Book[1000];
}
class Book
{
    int id;
    int InLibrary(Library Lib)
    {
        for (int i = 0; i < 1000; i++)
            if (id == Lib.books[i].id) return 1;
            return 0;
    }
}
```

При застосуванні останнього рефакторингу цей код перетвориться на наступний:

```
class Library
{
    Book[] books = new Book[1000];
    int InLibrary(Book book)
    {
        for (int i = 0; i < 1000; i++)
            if (book.id == books[i].id) return 1;
            return 0;
```

```
    }
}
class Book
{
    public int id;
}
```

Перейменування. При виконанні операції «Перейменування» підсистема оптимізації виконує операцію перейменування, що призначена конкретно кожного символу коду.

Перейменовувати можна назви полів, локальних змінних, методів, просторів імен, властивостей, типів, ідентифікаторів.

У випадку перейменування члена, який реалізує /перевизначає чи який реалізується /перевизначається членами інших типів, Visual Studio відкриває діалогове вікно, в якому говориться, про те, що виконання операції призведе до каскадного перейменування.

Для виконання операції перейменування стоячи на імені, яке слід перейменувати натискаємо праву кнопку миші (див. рис. 1).

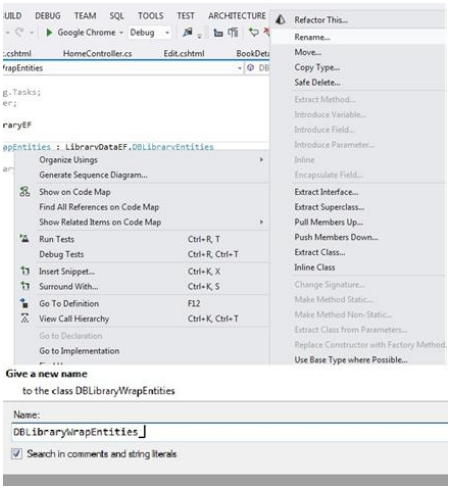

Рисунок 1. Перейменування

Переіменування методом смарт-тегів. Почніть змінювати ім’я (додавати чи видаляти символи) при цьому з’явиться підказка з пропозицією перейменування (див. рис. 2).

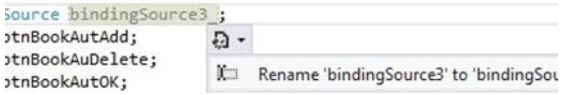

Рисунок 2. Перейменування методом смарт-тегів

Виділення методу полягає у виділенні з довгого чи важкого для сприйняття коду окремих його фрагментів і перетворенні їх в окремі методи (функції). Новий, виділений метод містить виділений код, а виділений код в існуючому елементі замінюється на виклик нового методу.

Для використання виділяємо потрібний рядок (рядки) та натискаємо праву кнопку миші (див. рис. 3-5).

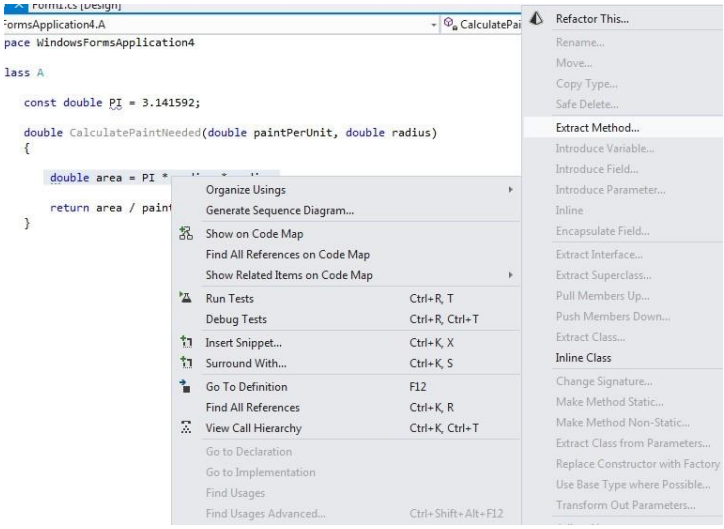

Рисунок 3. Виділення методу

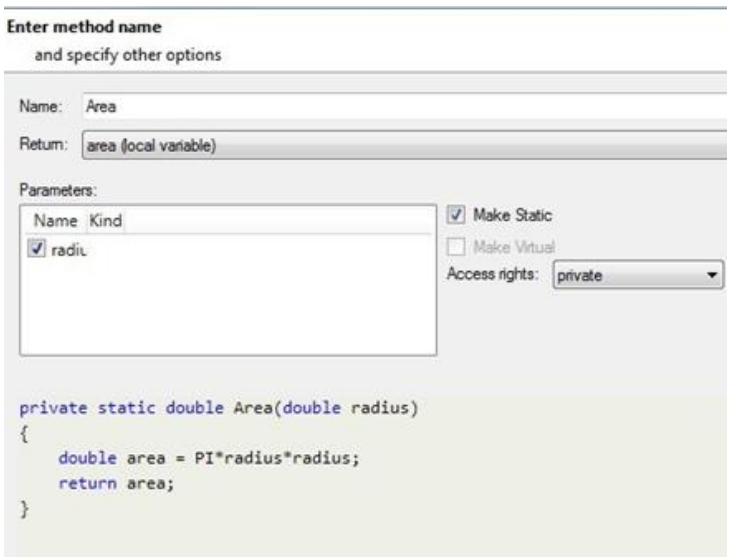

Рисунок 4. Виділення методу

```
class A
{
    const double PI = 3.141592;

    double CalculatePaintNeeded(double paintPerUnit, 
    {
        var area = Area(radius);

        return area / paintPerUnit;
    }

    private static double Area(double radius)
    {
        double area = PI*radius*radius;
```

Рисунок 5. Виділення методу

Інкапсуляція поля – дозволяє створювати з існуючого поля властивість, а потів послідовно оновити код, включивши до нього посилання на нову властивість. Якщо в класі є відкрите поле, то необхідно зробити його закритим і забезпечити методи доступу.

Якщо поле `public`, то при інкапсуляції поля модифікатор доступу замінюється на `private`, а потім генеруються методи доступу `get` та `set`. Стоячи на визначенні потрібного поля напишніть праву кнопку миші (див. рис. 6-9).

```
class Square
{
    public int width, height;
}
class MainClass
{
    public static void Main()
    {
        Square mySquare = new Square();
        mySquare.width = 110;
        mySquare.height = 150;
        // Output values for width and height.
        Console.WriteLine("width = {0}", mySquar
```

Рисунок 6. Інкапсуляція поля

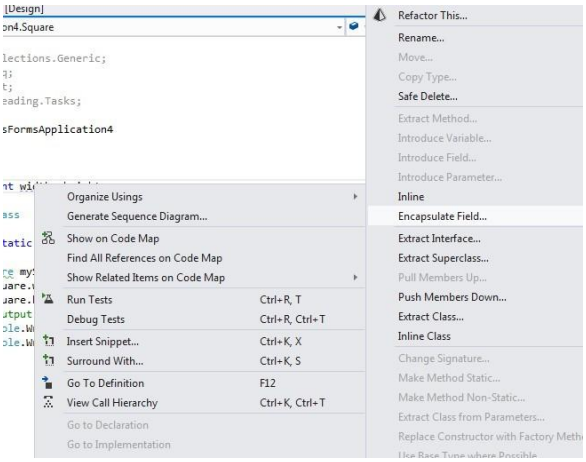

Рисунок 7. Інкапсуляція поля

Рисунок 8. Інкапсуляція поля

```
class Square
{
    private int width;
    public int height;

    public int Width
    {
        get { return width; }
        set { width = value; }
    }
}
class MainClass
{
    public static void Main()
    {
        Square mySquare = new Square();
        mySquare.Width = 110;
        mySquare.height = 150;
        // Output values for width and height
```

Рисунок 9. Інкапсуляція поля

Виділення інтерфейсу – якщо декілька клієнтів використовують одну підмножину інтерфейсу класу або в кількох класах певна частина інтерфейсу спільна, то рефакторинг забезпечує виділення цієї підмножини в окремий інтерфейс. При виконанні цієї операції в новому файлі генерується інтерфейс і курсор поміщається на початок цього файлу (див. рис. 10-13).

```
class A
{
    public void Method1(string s
    public void Method2(string s
    public void Method3(string s
}

class B
{
    public void Method1(string s
    public void Method2(string s
```

Рисунок 10. Виділення інтерфейсу

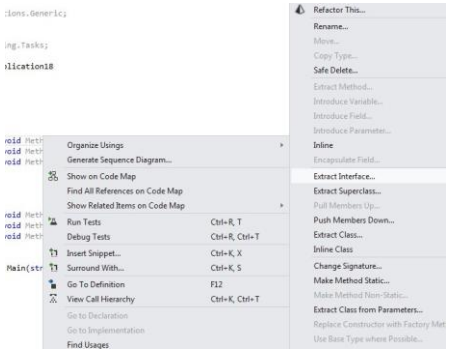

Рисунок 11. Виділення інтерфейсу

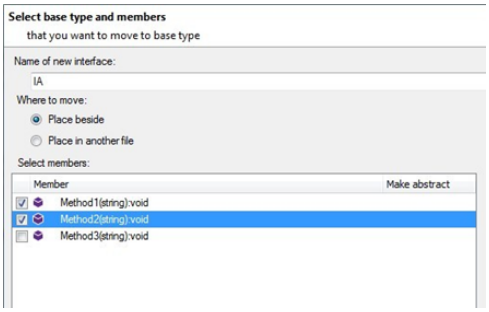

Рисунок 12. Виділення інтерфейсу

```
private interface IA
{
    void Method1(string s);
    void Method2(string s);
}

class A : IA
{
    public void Method1(string
    public void Method2(string
    public void Method3(string
}

class B
{
```

Рисунок 13. Виділення інтерфейсу

Зміна сигнатури — забезпечує простий спосіб видалення параметрів з методів. Ця операція змінює оголошення; в довільних місцях, де викликається член, параметр видаляється (див. рис. 14-16).

```
class A
{
    // Invoke on 'A'.
    public A(string s, int
}

class B
{
    void C()
    {
        // Invoke on 'A'.
```

Рисунок 14. Зміна сигнатури

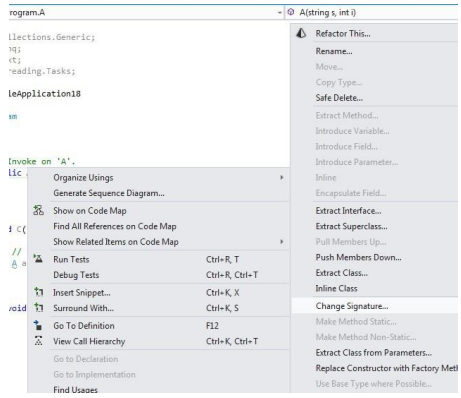

Рисунок 15. Зміна сигнатури

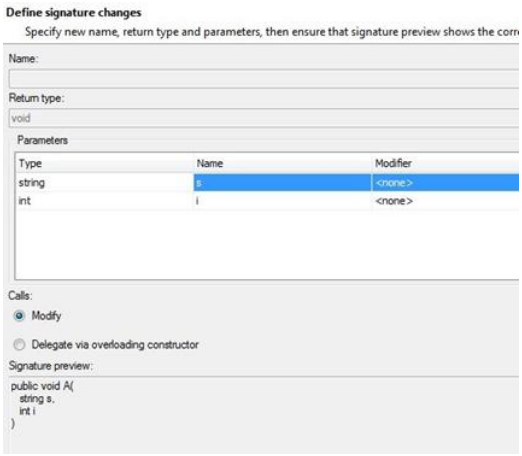

Рисунок 16. Зміна сигнатури

ДОДАТОК Б. ПУБЛІКАЦІЇ ЗА РЕЗУЛЬТАТАМИ ДОСЛІДЖЕННЯ

ВИМОГИ ДО КАДРОВОГО ЗАБЕЗПЕЧЕННЯ РИНКУ ПРАЦІ У ГАЛУЗІ DEVOPS

Автор: Панчук І.М., магістр
Науковий керівник: Чикунов П.О., к.т.н., доц.
Бахмутський навчально-науковий професійно-педагогічний інститут ХНУ імені В. Н. Каразіна

У сучасних умовах розробки програмного забезпечення якісного коду стало критичним фактором для успіху бізнесу. Рефакторинг коду є однією з найважливіших практик у DevOps, яка дозволяє знижувати технічний борг і підвищувати ефективність роботи системи.

Далі наведено результати аналізу вимог до кадрового забезпечення підготовки фахівців DevOps, які будуть займатися рефакторингом програмного коду, під час проведення якого визначено ключові компетенції та навички, необхідні для цього процесу.

DevOps – це методологія розробки програмного забезпечення, яка поєднує в собі розробку (Development) і операційні процеси (Operations) для швидшої доставки якісного програмного продукту.

Рефакторинг – це процес поліпшення структури існуючого коду без зміни його функціональності для підвищення продуктивності та зручності підтримки. Рефакторинг важливий для зниження технічного боргу і забезпечення адаптивності системи до майбутніх змін [1].

До основних технічних навичок фахівців DevOps, необхідних для рефакторингу коду, можна віднести такі.

По-перше, глибоке розуміння сучасних мов програмування, таких як Python, Java, C#, JavaScript, які найчастіше використовуються у DevOps, знання інструментів автоматизації, таких як Jenkins, GitLab CI/CD, Ansible, що допомагають прискорити процеси розробки та розгортання програмної системи, уміння працювати з контейнерними технологіями (Docker, Kubernetes) для управління середовищем розробки та тестування тощо.

По-друге, наявність софт-скілів та навичок командної роботи: здатність ефективно взаємодіяти з командою розробників, аналітиків і менеджерів проектів, готовність до швидкого реагування на зміни у вимогах проекту та вміння приймати рішення в умовах невизначеності, вміння виявляти та усувати дефекти коду в процесі рефакторингу.

По-третє, знання принципів та інструментів DevOps, зокрема розуміння основних принципів інтеграції та безперервного розгортання для оптимізації життєвого циклу розробки (Continuous Integration/Continuous Deployment), використання найкращих практик та інструментів для моніторингу, логування і тестування, таких як Prometheus, Grafana, ELK Stack.

До світних вимог підготовки фахівців DevOps можна віднести необхідність формальних знань у сфері комп'ютерних наук, програмування та IT інфраструктури, та обов’язкове проходження протягом життя спеціалізованих

курсів та сертифікації з DevOps та методів рефакторингу, зокрема курсів AWS, Google Cloud, сертифікації Docker та Kubernetes.

Також конче важливо забезпечити діючим фахівцям участь у практичних стажуваннях та навчальних програмах, орієнтованих на DevOps.

У розвитку навичок рефакторингу та інших ключових компетенцій складно переоцінити роль менторства та підтримки молодих спеціалістів.

Основними викликами та перспективами кадрового забезпечення в DevOps є високі вимоги до технічних навичок та досвіду фахівців DevOps, та недостатність фахівців з глибокими знаннями у сфері рефакторингу та забезпечення якості коду.

Для подолання викликів необхідно запровадження програм професійного розвитку та навчання всередині компаній.

Трендами та перспективи розвитку кадрового ринку DevOps та рефакторингу є зростання попиту на фахівців, здатних швидко адаптуватися до змін у технологічному середовищі, зокрема акцент на інтеграцію штучного інтелекту та машинного навчання у процеси DevOps для автоматизації рефакторингу та інших процесів.

Для навчальних закладів, які займаються підготовкою та перепідготовкою фахівців DevOps та рефакторингу, можна надати рекомендацію розробки комплексних програм навчання з акцентом на практичні навички та командну роботу, з урахуванням обов'язкової співпраці між бізнесом і освітніми установами для забезпечення актуальних навчальних матеріалів і практик.

Висновки. Аналіз ключових аспектів кадрового забезпечення ринку DevOps та рефакторингу показав таке:

– необхідність наявності висококваліфікованих кадрів з технічними і софт-скілами, які відповідають вимогам DevOps та рефакторингу;

– важливість постійного розвитку та навчання фахівців для підвищення їхньої конкурентоспроможності на ринку праці.

Список використаних джерел

1. Чикунов П.О. Рефакторинг коду при конструюванні програмного забезпечення // Актуальні тенденції розвитку освіти, науки та технологій : Матеріали VI Міжн. наук.-практ. конф. (м. Бахмут, м. Харків, 18 травня 2023 р.). : у 2-х ч. / За заг. ред. Г. Г. Михальченко. Бахмут – Харків: ННІПІ УІПА, 2023. Ч 1. – С. 98-100