Міністерство освіти і науки України
Національний технічний університет «Харківський політехнічний інститут»

На правах рукопису

НАГОРНИЙ КОСТЯНТИН АНАТОЛІЙОВИЧ

УДК 681.5: 658.512

МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ПОСТ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ

Спеціальність 05.13.06 – інформаційні технології

Дисертація на здобуття наукового ступеню
кандидата технічних наук

Науковий керівник
доктор технічних наук, професор
Ткачук Микола Вячеславович

Харків – 2016

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 5

ВСТУП 7

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМ СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ 14

1.1. Деякі критичні проблеми етапу супроводу програмних систем (ПС) на основі застосування об'єктно-орієнтованого підходу (ООП) 15

1.2. Актуальність вирішення проблеми наскрізної функціональності при супроводі успадкованих ПС (УПС) 20

1.3. Причини виникнення та стисла характеристика основних пост об'єктно-орієнтованих технологій (ПООТ) розробки програмного забезпечення (ПЗ) 26

1.4. Постановка задачі розробки та дослідження модельно-технологічного інструментарію для оцінки ефективності застосування ПООТ при супроводі УПС 35

Висновки по розділу 1 37

РОЗДІЛ 2. МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ МОДЕЛЬНО-ТЕХНОЛОГІЧНОГО ІНСТРУМЕНТАРІЮ ДЛЯ ОІІНКИ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ПОСТ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ 38

2.1. Формалізація задачі визначення ефективності застосування пост об'єктно-орієнтованих технологій у процесах супроводу УПС і знання-орієнтований підхід до її вирішення 39

2.2. Аналіз і мотивований вибір експертних методів та кількісних метрик для оцінки структурної складності УПС 43

2.3. Логіко-лінгвістичний підхід до моделювання та аналізу стану вимог до УПС в процесі її супроводу 52

2.4. Побудова архітектурних моделей для аналізу технологічних особливостей застосування ПООТ 63

2.5. Розробка метрик для комплексної оцінки рівня присутності наскрізної функціональності (НФ) у програмному коді УПС 73

Висновки по розділу 2 79

РОЗДІЛ 3. РОЗРОБКА МОДЕЛЕЙ, МЕТОДІВ ТА ТЕХНОЛОГІЧНИХ ПРОЦЕДУР ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ПОСТ ОБЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ 81

3.1. Розробка алгоритмічної моделі процесу експертної оцінки ефективності застосування ПООТ 82

3.2. Розробка метода визначення стану УПС у процесі супроводу 84

3.3 Розробка метрик та алгоритмів визначення питомих витрат на застосування ПООТ на основі їх архітектурних моделей 91

3.4. Процедура комплексної оцінки ефективності використання ПООТ із застосуванням методів нечіткої логіки 99

3.5. Загальна схема прикладної інформаційної технології для реалізації запропонованого підходу 107

Висновки по розділу 3 113

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ МОДЕЛЕЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ 114

4.1. Опис предметної області для задачі вибору ПООТ в процесі супроводу УПС 114

4.2. Розробка архітектури та основних програмних компонентів інструментального CASE-засобу для реалізації запропонованого підходу 119

4.3. Методика проведення експерименту, вхідні дані та приклади розрахунків обчислювальних експериментів 132

4.4. Аналіз отриманих результатів і практичні рекомендації по використанню ПООТ для супроводу УПС 142

Висновки по розділу 4 148

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 149

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 152

ДОДАТОК А. АКТИ ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ 171

ДОДАТОК Б. ПРИКЛАДИ ДОКУМЕНТАЦІЇ ПО РОЗРОБЦІ ІНСТРУМЕНТАЛЬНОГО ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ПООТ 177

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

АОП – аспектно-орієнтований підхід

БД – база даних

БЛ – бізнес-логіка

ЖЦ – життєвий цикл

ІС – інформаційна система

КОП – контекстно-орієнтований підхід

КПР – компонентне програмне рішення

ЛАП – лінійка архітектурних примітивів

ЛПП – лінійка програмних продуктів

НФ – наскрізна функціональність

МАІ – метод аналізу ієрархій

МД – модель даних

ООП – об’єктно-орієнтований підхід

ПЗ – програмне забезпечення

ПООТ – пост об’єктно-орієнтований підхід

ПрО – предметна область

ПС – програмна система

РАП – розширений архітектурний примітив

СКБД – система керування БД

СУВ – система управління вимогами

УПС – успадкована програмна система

ФВ – функціональні вимоги

ФОП – функціонально-орієнтований підхід

AM – algorithmic model

AOSD – aspect-oriented software development

API – application programming interface

CASE – computer-aided software engineering

CF – crosscutting functionality

CFR – crosscutting functionality ratio

CFL – crosscutting functionality level

COSD – context--oriented software development

SC – structural complexity

DS – degree of scattering

IDE – integrated development environment

IDEF – Icam DEFinition

GUI – graphical user interface

RMS – requirements management system

RUP – Rational Unified Process

SoC – separation of concerns

SRS – software requirement specification

UML – Unified Modeling Language

ВСТУП

Актуальність теми. Парадигма використання об’єктно-орієнтованого підходу (ООП) для розробки програмних систем (ПС) передбачає, що вони розглядаються як сукупність програмних сутностей, які відображають характеристики та особливості взаємодії інформаційних об’єктів у деякій предметній області (ПрО). Застосування ООП передбачає можливість локалізації будь-якої бізнес-логіки в окремих класах ПЗ, що має на меті також забезпечити можливість гнучкого внесення змін у вихідний код, якщо цього потребують зміни у вимогах користувачів до ПС. Саме тому парадигма використання ООП, яка була запропонована та розвинута в роботах таких визнаних фахівців як Г. Буч (G. Booch), І. Якобсон (I. Jacobson), Дж. Рамбо (J. Rambaugh), Дж. Оделл (J. Odell) та інші, є досить ефективною для розробки та супроводу значної кількості ПС у різних ПрО.

В той же час, останні тенденції в області розробки та супроводу сучасних ПС, зокрема, такі як їх постійно зростаючі структурна складність та інтенсивність внесення змін у вимоги користувачів, призвели до того, що у практиці застосування ООП проявилися певні кризові явища. Одне із них отримало назву феномену наскрізної функціональності (crosscutting functionality) – НФ, суть якої полягає в неможливості виділити програмний код НФ в окремий програмний клас, натомість код НФ є розсіяним з-поміж класів, в яких реалізовані інші вимоги до ПС. Для подолання негативних наслідків НФ були запропоновані нові технології розробки ПЗ, що розширюють можливості застосування ООП за рахунок вбудованих механізмів, які дозволяють ізолювати програмний код НФ в окремому програмному модулі, та потім у не інвазійний спосіб отримати нову конфігурацію ПС. Ці нові підходи до створення ПС можуть бути узагальнено визначені терміном пост об’єктно-орієнтовані технології (ПООТ), і проблеми їх розробки та застосування досліджуються в роботах таких відомих вітчизняних та закордонних

вчених як О. В. Палагін, І. В. Сергієнко, Д. В. Федасюк, І. Соммервіль (I. Sommerville), С. Апель (S. Apel), К. Поль (K. Pohl), Л. Аверсано (L. Aversano), Е Фігуредо (E. Figueiredo) та інших фахівців.

Слід зазначити, що практичне застосування ПООТ, зокрема, в процесах супроводу успадкованих ПС (УПС), які були розроблені на основі ООП, пов'язане з додатковими витратами часу та інших проектних ресурсів. Тому виникає проблема визначення ефективності використання певної ПООТ для проведення модернізації УПС із урахуванням особливостей її функціональності та впливу зовнішнього середовища у вигляді змін вимог користувачів. Таким чином, постановка задачі дослідження цієї роботи є досить актуальною та практично значущою.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалася на кафедрі програмної інженерії та інформаційних технологій управління НТУ «Харківський політехнічний інститут» відповідно до завдань прикладної держбюджетної НДР МОН України: «Розробка інтелектуальних моделей та технологій для підвищення ефективності проектування та супроводу складних програмних систем» (ДР № 0111U002288), де здобувач брав участь як співвиконавець окремих розділів.

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення ефективності використання пост об'єктно-орієнтованих технологій (ПООТ) у процесі супроводу УПС шляхом розробки та застосування знання-орієнтованих інформаційних моделей, методів та інструментальних засобів. Для досягнення цієї мети в роботі поставлені та вирішені такі задачі:

– розглянуті особливості процесу супроводу та структурної модифікації УПС, які розроблені на основі ООП, із урахуванням проблем, що виникають при появі ефекту наскрізної функціональності (НФ);

– проведено порівняльній аналіз основних існуючих ПООТ, а саме: аспектно-орієнтованого підходу (АОП), функціонально-орієнтованого підходу (ФОП) та

контекстно-орієнтованого підходу (КОП), з метою вивчення особливостей структурних механізмів локалізації та усунення негативних проявів НФ у УПС;

– запропоновано знання-орієнтований методологічний підхід для комплексної оцінки ефективності застосування ПООТ в процесах розробки та супроводу ПС;

– розроблена алгоритмічна модель процесу оцінки ефективності застосування ПООТ;

– запропонована модель багатовимірного інформаційного простору для оцінки стану УПС в процесі їх супроводу та визначення характеристик ПООТ;

– розроблені архітектурні моделі окремих ПООТ для визначення питомих витрат при їх застосуванні для структурної модифікації УПС;

– запропоновані кількісні метрики для оцінки загального рівня присутності та ступеня розповсюдження НФ у вихідному коді УПС;

– розроблений нечіткий підхід до визначення кінцевої ефективності застосування окремих ПООТ при супроводі УПС із урахуванням слабоформалізованого та багатовимірного характеру діючих чинників та критеріїв їх оцінки;

– реалізовано основні програмні компоненті прикладної інформаційної технології для практичного застосування запропонованих моделей та інструментальних засобів;

– проведено експериментальне дослідження запропонованого підходу та на основі аналізу отриманих результатів надані практичні рекомендації щодо підвищення ефективності використання ПООТ в процесах супроводу УПС.

Об'єктом дослідження є процеси супроводу успадкованих програмних систем (УПС), розроблених на основі об'єктно-орієнтованого підходу.

Предметом дослідження є моделі, методи та інструментальні засоби оцінки ефективності застосування ПООТ в процесах супроводу УПС.

Методи дослідження базуються на застосуванні принципів сучасної програмної інженерії, зокрема на використанні кількісних метрик якості ПЗ,

об’єктно-орієнтованих та пост об’єктно-орієнтованих методах аналізу та синтезу ПЗ; математичного апарату теорії множин; експертних методів теорії прийняття рішень та нечіткої логіки; уніфікованої мови моделювання UML (Unified Modeling Language) та мови опису програмних архітектур ADL (Architecture Description Language) для аналізу та синтезу програмних рішень.

Наукова новизна отриманих результатів:

– *вперше:* запропонована алгоритмічна модель процесу визначення ефективності застосування пост об'єктно-орієнтованих технологій (ПООТ) при супроводі успадкованих програмних систем (УПС), яка відрізняється від існуючих підходів комплексним використанням багатовимірного інформаційного ресурсу, експертних методів та відповідних метрик, що дозволяє кількісно оцінити ступінь ефективності застосування окремих ПООТ у залежності від суттєвих характеристик наявної УПС;

– *отримали подальший розвиток:* методи та інструментальні засоби аналізу та визначення стану УПС у процесі її супроводу шляхом використання сукупності обчислювальних алгоритмів і кількісних критеріїв, що дозволяє враховувати як структурну складність так і динаміку змін у вимогах до певної УПС в процесі її супроводу;

– *отримали подальший розвиток:* моделі та інструментальні засоби дослідження технологічних особливостей використання ПООТ в процесах супроводу УПС шляхом розробки еталонних архітектурних моделей окремих ПООТ, що дає можливість кількісно визначити питомі витрати при їх застосуванні для програмної модифікації УПС з метою усунення негативних наслідків присутності НФ;

– *удосконалено:* інформаційна технологія оцінки ефективності застосування ПООТ в процесах супроводу УПС за рахунок використання методів нечіткого виводу у поєднанні із кількісними метриками оцінки як загального рівня присутності, так і ступеня розповсюдженості НФ в програмному коді УПС, що

забезпечує можливість попереднього аналізу та оцінки ефективності альтернативних варіантів проведення модернізації успадкованих програмних систем.

Практичне значення отриманих результатів визначається тим, що розроблені в дисертаційній роботі моделі, методи та інструментальні засоби можуть бути використані для підвищення ефективності процесів супроводу успадкованих програмних систем із застосуванням як вже існуючих так і перспективних ПООТ. Запропонований комплексний підхід до оцінки ефективності використання ПООТ і розроблені для його підтримки інструментальні програмні засоби були успішно використані для вирішення цих завдань на прикладі застосування аспектно-орієнтованої технології в розробках по прикладній держбюджетній НДР МОН України, при виконанні проектів в компаніях Bit media e-learning solution GmbH & Co KG (м. Грац, Австрія) та ТОВ «ІНТЕРПАК ІНФОРМАЦІЙНІ СИСТЕМИ» (м. Харків), а також впроваджені в навчальному процесі кафедри програмної інженерії та інформаційних технологій управління НТУ «ХПІ» в дисциплінах «Основи проектування ПЗ», «Аналіз вимог до ПЗ», «Методи та засоби автоматизації процесів життєвого циклу ПЗ».

Особистий внесок здобувача. Усі наукові результати, викладені в дисертації, отримані здобувачем особисто. У роботах, які написано у співавторстві, особистий внесок здобувача полягає у такому: запропонована схема структурної адаптації УПС із використанням ПООТ [25]; виконаний порівняльний аналіз ПООТ та розроблена структура багатовимірного інформаційного простору для оцінки ефективності окремих ПООТ [46]; запропоновано логіко-лінгвістичний підхід до оцінки стану УПС у процесі їх супроводу, який враховує як структурну складність так і динаміку змін у вимогах користувачів таких систем [153]; розроблені архітектурні моделі окремих ПООТ і методика оцінки питомих витрат при їх застосуванні для структурної адаптації УПС з метою усунення проблем НФ [123]; запропонований підхід із використанням методів нечіткого виводу для

оцінки ефективності альтернативних варіантів проведення модернізації УПС на основі ПООТ [147]; досліджені особливості застосування аспектно-орієнтованого підходу для супроводу УПС [68]; розглянуті особливості застосування функціонально-орієнтованого підходу для супроводу УПС [76]; досліджені особливості застосування контекстно-орієнтованого підходу для супроводу УПС [80]; розроблена архітектура інструментального CASE-засобу для дослідження ефективності використання ПООТ [166]; розроблена методика експериментального дослідження ефективності використання ПООТ [160], побудовані метрики оцінки рівня присутності та характеру розповсюдження НФ у програмному коді УПС [86]; запропонована алгоритмічна модель процесу загальної оцінки ефективності використання ПООТ при супроводі УПС [150].

Апробація результатів дисертації. Основні положення та результати досліджень пройшли апробацію та одержали позитивну оцінку на XI-й Міжнародній науково-технічній конференції «Системний аналіз и інформаційні технології» (Київ, 2009); на VII-й Міжнародній науково-технічній конференції «УкрПРОГ-2010» (Київ, 2010); на XVII, XIX, XXII Міжнародних науково-практичних конференціях «Інформаційні технології: наука, техніка, технологія, освіта, здоров'я:», (Харків - 2009, 2011, 2014); на 11th International Conference on ICT in Education, Research and Industrial Applications: ICTERI-2015 (Львів, 2015), а також на наукових семінарах кафедри програмної інженерії та інформаційних технологій управління НТУ «ХПІ» і кафедри комп'ютерних систем і мереж Національного аерокосмічного університету імені М. Є. Жуковського «Харківський авіаційний інститут».

Публікації. За результатами дисертаційного дослідження опубліковано 12 наукових праць, у тому числі 5 статей у наукових фахових виданнях України з технічних наук, з них 3 публікації у виданнях, що входять до наукометричних баз *"Index Copernicus", "Ultrich's Periodicals Directory"*, 1 стаття – у виданні "Springer", 6 публікацій у тезах та матеріалах міжнародних конференцій (у тому

числі 2 публікації у виданнях, що входять до науковометричної бази *“Scopus”*). Зазначені публікації повністю відображають наукові результати й висновки дисертаційної роботи.

Структура та обсяг дисертації. Дисертація складається із вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 182 сторінки, у тому числі: основний текст дисертації на 151 сторінці, містить 42 рисунка та 19 таблиць, перелік інформаційних джерел із 173 найменувань на 19 сторінках та 2 додатка на 12 сторінках.

РОЗДІЛ 1.
АНАЛІЗ ПРОБЛЕМ СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

В даному розділі розглянуті деякі сучасні проблеми супроводу програмних систем (ПС), які розробляються та супроводжуються на основі застосування об'єктно-орієнтованої парадигми (ООП) програмування та ті наслідки, до яких призводить поява в них феномену наскрізної функціональності (НФ). Показані причини виникнення пост об'єктно-орієнтованих технологій (ПООТ) розробки ПС, як засобу для усунення негативних проявів появи та присутності НФ у програмному коді такого важливого типу ПС як успадковані програмні системи (УПС), сформульовані мета і основні задачі даного дисертаційного дослідження.

В першому підрозділі, на основі аналізу сучасних інформаційних джерел розглянуті стан і основні напрямки сучасних досліджень в галузі супроводу ПЗ на основі використання ООП.

В другому підрозділі мотивована актуальність вирішення проблеми усунення негативних наслідків появи НФ при супроводі УПС.

У третьому підрозділі проаналізовано причини виникнення та надана стисла характеристика основних ПООТ для розробки програмного забезпечення, які мають механізми для усунення проблеми НФ, та проведено порівняльний аналіз їх основних якісних ознак

В четвертому підрозділі сформульовані загальна мета та основні задачі даного дисертаційного дослідження.

1.1. Деякі критичні проблеми етапу супроводу програмних систем (ПС) на основі застосування об’єктно-орієнтованого підходу (ООП)

У стандарті «Життєвий цикл програмних систем» ISO/IEC 12207:2008 (“Systems and software engineering – Software Life Cycle Process”) [1] зазначено, що процес супроводу будь-якої ПС є одним із головних етапів її життєвого циклу (ЖЦ), який розпочинається під час експлуатації ПС користувачами, та має бути спрямованим на виявлення та виправлення помилок у програмному коді, на додання нових функцій ПС, пов’язаних з виникненням нових вимог, та постійної її адаптації до існуючих умов експлуатації.

На рис. 1.1 представлено загальну схему основних етапів процесу супроводу ПС згідно з міжнародним стандартом Життєвий цикл програмних систем – супровід» ISO/IEC14764-2006 (“Software Engineering — Software Life Cycle Processes — Maintenance) [2].

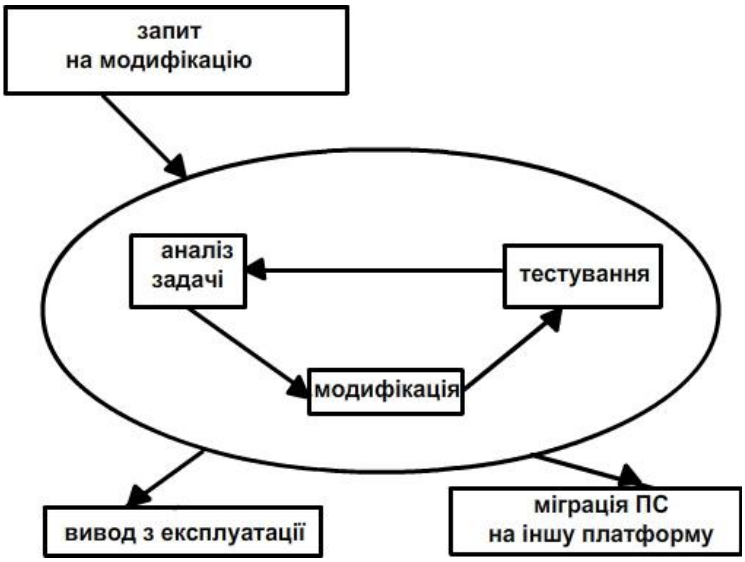


Рисунок 1.1 – Основні активності процесу супроводу ПС,
згідно із стандартом ISO/IEC14764-2006

Основними етапами процесу супроводу (див. рис. 1.1) є аналіз задачі (тобто запиту на модифікацію ПС), моделювання варіантів її реалізації, модифікація ПС, та тестування. Зовнішній запит на модифікацію (modification request) – це будь-які задокументовані помилки або нові вимоги, що надходить від кінцевих користувачів в процесі активної експлуатації ПС. В подальшому кожний такий запит може бути класифікований (ідентифікований) як коригуючий – це виправлення помилок у програмному коді ПС, або доповнюючий – це коли мається на увазі розширення наявної функціональності ПС [1, 2].

Протягом процесу супроводу, для адаптації ПС до постійно змінюваного навколишнього середовища, може бути прийняте рішення щодо міграції ПС на більш сучасну платформу (робоче середовище), або про необхідність повного виводу ПС з експлуатації. Ці проблеми та відповідні підходи до їх вирішення більш докладно розглядаються в інших роботах та виходять за рамки даного дослідження. За різними оцінками, трудомісткість фази супроводу займає більше 50% всіх витраті життєвого циклу ПС, тобто є досить витратною та важливою [3].

Слід зазначити, що за роки активної експлуатації ПС вона постійно модифікується і доповнюється новими функціональними модулями, в ній накопичується велика кількість оброблених даних та знань, що є критичними для бізнес-процесів компанії. Також накопичується досвід роботи з ПС, який має цінність для її кінцевих користувачів, не зважаючи на те, що технології реалізації ПС за цей час, як правило, морально застарівають. Ці фактори ускладнюють використання нових технологій у діючий ПС в процесі її супроводу і унеможливлюють перехід користувачів на більш нові версії системи. Такі програмні системи в літературі називаються успадкованими (legacy system) [4, 5]. Успадковані програмні системи (УПС) мають ряд особливостей, що ще більше ускладнюють процес їх супроводу, а саме [6]:

– команда фахівців із супроводу УПС не приймала участі в первинній розробці цієї ПС, внаслідок цього зростає вірогідність некоректного використання відповідних програмних засобів та технологій;
– не задокументовані зміни до первинної архітектури ПС, що були внесені на стадії розробки;
– проектна документація є застарілою, або відсутньою взагалі;

та деякі інші.

За два останніх десятиріччя отримала активний розвиток ООП розробки ПЗ, про що свідчить велика кількість мов програмування, що її реалізують, таких як перевірені часом: C++ [7], C# [8], Java [9], Objective-C [10] та інші, а також нові, що з'явилися за останню декаду, наприклад D [11] , X++ [12], Swift [13] та ін. Про популярність об'єктно-орієнтованих мов програмування свідчить і наявність їх версій для багатьох платформ (стаціонарних та мобільних) та операційних систем (ОС), наприклад Windows [14], UNIX-базовані ОС [15], операційних систем компанії Apple [16, 17], Android [18] і т.д. Отже, можна мотивовано стверджувати, що велика частина УПС, які з'явилися за цей період часу і наразі знаходиться в процесі їх супроводу, створені саме з використанням ООП - засобів, тому для таких УПС є актуальними як загальні проблеми процесів супроводу, зазначені вище, так і певні специфічні проблеми ООП - підходу в цілому, що спричинені необхідністю адаптації ПС до зовнішнього середовища (див. напр. у [4]).

Головна проблема ООП – підходу полягає в тому, що для того, щоб задовольнити постійні запроси на модифікацію (виправлення помилок та реалізація новою функціональності), в об'єктно-орієнтованій ПС потрібно постійно проводити модифікацію її структуру на рівні програмних класів та зв'язки між ними. Стандартним способом для цього в ООП є розробка нових (додаткових) класів та модифікація зв'язків між новими та старими програмними модулями, зміни в дереві наслідування, реалізації нових та модифікація вже застосованих шаблонів або патернів проектування (design pattern), таких як,

наприклад, шаблони колекції GoF (Gang-of-Four) [19]. Постійна модифікація програмного коду ПС призводить до таких проблем як:

- збільшення ступеня зв'язності коду (code coupling) між окремими програмними класами [20],
- розростання глибини дерева наслідування (depth of inheritance) властивостей класів [21, 22],
- некоректна модифікація раніше реалізованих та некоректна реалізація нових проектних патернів [23],
- поява у вихідному коді зразків анті-патернів (anti-patterns) [24]

та деякі інші.

Таким чином, зазначені вище недоліки ООП вказують на те, що процес супроводу обох типів систем у певній бізнес-організації або на підприємстві, а саме:

(I) ПС, які безпосередньо розроблені та супроводжуються однією та тією же групою виконавців,

(II) УПС, які були розроблені раніше, а тепер супроводжуються іншою командою фахівців,

є вельми актуальною та достатньо складною науково-технічною задачею.

В [25] розглядається актуальність використання в процесі супроводу УПС сучасних адаптивних підходів до створення ПЗ, які дозволять більш гнучко реагувати на запити на модифікацію як ПС так і УПС, тобто створювати так зване «життєздатне» ПЗ (*viable software*) [26], шляхом впровадження та застосування принципів "програмної кібернетики" (*software cybernetics*) [27]. Вони передбачають використання в процесі розробки та супроводу ПЗ загальних схем систем управління із зворотним зв'язком та застосування кількісних характеристик та відповідних метрик для оцінки якості роботи ПС.

В [25] зазначено, що задача структурної адаптації обох типів систем (I) –(II) у неформалізованому вигляді може бути сформульована наступним чином:

як при змінах деяких факторів навколишнього середовища, в якому використовується ПС (УПС), потрібно проводити її модифікацію з метою забезпечення необхідних значень визначеного критерію якості функціонування цієї системи?

Більш детально ця постановка проблеми структурної адаптації передбачає формулювання наступних чотирьох тверджень [25]:

1) найбільш важливим типом збурень, що впливають на будь-яку ПС або УПС у зовнішньому середовищі її функціонування, слід вважати вимоги користувачів (user requirements), які відображають собою потрібні властивості цільової ПС, тобто як функціональні так і нефункціональні вимоги, або атрибути якості [4];

2) загальну схему реалізації процедур побудови адаптивної структури ПС (УПС), реалізованої з використанням ООП, можливо розглядати як *схему управління зі зворотнім зв'язком* [28], тобто застосовувати кібернетичний підхід до структурної адаптації;

3) у якості *управляючого впливу* для структурної адаптації доцільно обрати технології програмування, які базуються на ООП, та мають при цьому додаткові механізми усунення наявних недоліків ООП-підходу;

4) критерієм якості управління процесом структурної адаптації необхідно вибрати такий показник якості ПЗ, який би не залежав від предметної області розробки або застосування ПС (УПС).

У якості саме такого критерію якості процесу структурної адаптації доцільно обрати такий універсальний показник як *супроводжуваність програмного забезпечення* (software maintainability) [4], оскільки він є однаково важливим для ПЗ різного призначення: для систем організаційного управління (information management systems), систем реального часу та вбудованих систем (real-time and embedded systems), систем аналітичної обробки даних (analytical data processing systems) та ін.

Таким чином, для подолання основних недоліків ООП в процесі супроводу обох типів систем: (I) звичайних ПС, та (II) успадкованих ПС (УПС), необхідно мати можливість проводити адаптацію їх структури згідно з якісними твердженнями 1) – 4). В подальшому, у цьому дослідженні розглядаються проблеми структурної адаптації саме УПС, як більш складний та досить часто зустрічаючийся на практиці тип проектів по супроводу ПЗ.

1.2. Актуальність вирішення проблеми наскрізної функціональності при супроводі успадкованих ПС (УПС)

Вивченню критичних проблем сучасного етапу розвитку ООП (див. підрозділ 1.1), та пошуку шляхів подолання їх негативному впливу на процес супроводу УПС присвячена велика кількість досліджень (див. наприклад у [29, 30, 31, 32, 33]). При цьому в більшості цих робіт зазначається, що внаслідок високої інтенсивності появи запитів на модифікацію ПЗ, що спричинені високою динамікою змін у вимогах користувачів, у деяких УПС виникає феномен наскрізної функціональності (crosscutting functionality) [34]. Наскрізна функціональність (НФ) – це цілісний, на рівні опису вимог, запит кінцевого користувача до функціональності ПС, який розсіюється (crosscut) на рівні розробки ПЗ. Тобто, це така частина бізнес-логіки УПС, яка не може бути локалізована в окремий програмний модуль у вигляді вихідного коду, але залишається самостійною у вигляді вимоги до цієї УПС. Загальна схема виникнення та наслідки присутності НФ при супроводі УПС відображена на рис. 1.2. Графічні елементи на цій схемі умовно розподіляються на три групи:

1. Основні графічні елементи – вони представляють собою властивості УПС, специфікації вимог, компоненти програмної реалізації УПС, помилки та звіти про їх прояв.
2. Стрілки напрямку, що візуально розділені на три типи, а саме:

а) пунктирні стрілки відображають звичайні активності процесу супроводу,

б) хвилясті стрілки відображають проблемні активності у цьому процесі, тобто ті, що зумовлюють появу НФ;

в) точкові стрілки відображають активності-наслідки присутності НВ.

3. Елементи в пунктирних рамках відображають можливі вирішення негативних наслідків присутності проблеми НФ.

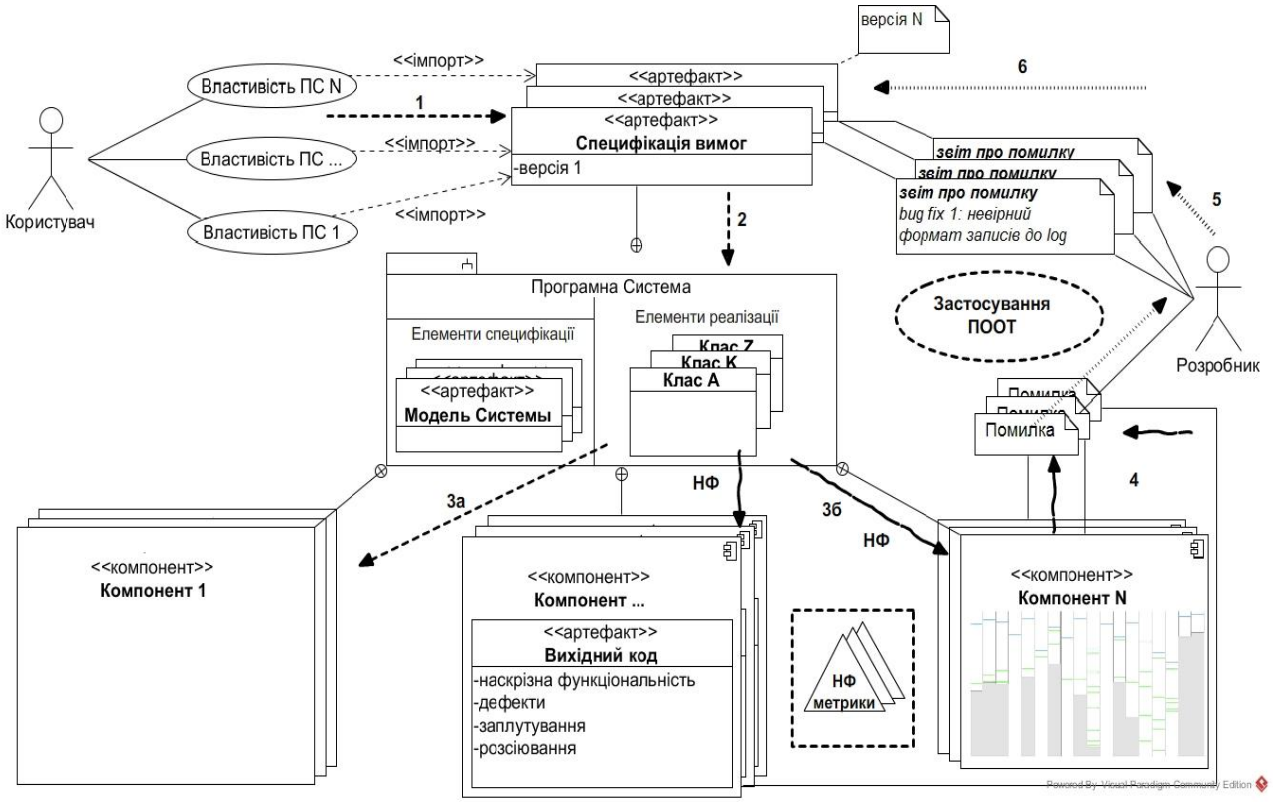


Рисунок 1.2 – Концептуальна схема причин виникнення НФ та її наслідки у процесі супроводу УПС

Процес виникнення та подальшого впливу НФ на УПС на рис. 1.2 умовно розділений на шість активностей, а саме:

1. Кінцеві користувачі УПС є джерелом запитів на модифікацію системи, що відображають нові вимоги на додання нових властивостей (надалі в тексті – функціональність) до неї. На основі вимог, що надійшли від користувачів,

розробляється поточна версія специфікації вимог (СВ), ця активність схематично зображений стрілкою «1».

2. Вимоги, що увійшли до СВ, трансформуються у відповідні моделі компонентів УПС (наприклад, у вигляді UML - діаграм) та знаходять відображення у відповідних артефактах програмної реалізації системи, тобто у вихідному код програмних класів УПС. Ця активність схематично зображена стрілкою «2».

3. Програмні класи групуються у програмні компоненти УПС, ця активність показана стрілками «3а» та «3б». Стрілка «3а» указує на компоненти, в яких НФ відсутня, а стрілки «3б», що відмічені додатково як «НФ», вказують на компоненти УПС, у вихідному коді яких була зареєстрована НФ. На схемі показані основні негативні фактори впливу НФ (що будуть деталізовані в цому розділі) та схематичне зображення НФ у вихідному коді системи (цей елемент також буде деталізований в цьому розділі). Таким чином, в програмному коді УПС присутні компоненти, які є «вільними» від НФ, а також такі, що є «ураженими» НФ.

4. НФ, в свою чергу, є джерелом програмних помилок у вихідному коді, що відображено стрілкою «4».

5. Такі помилки реєструються та відповідним чином формалізуються як звіти про помилки (bug report) у системі управління помилками (bug tracking system), такій як, наприклад, *Mantis bug tracker* [35]). Ця активність схематично позначена стрілкою «5».

6. Відповідні звіти про помилки знаходять відображення у новій версії специфікації вимог, що зображено стрілкою «6».

Таким чином, цикл управління супроводом УПС замикається і активності (1) – (6) повторюються знову.

У відповідності до сформульованих у підрозділі 1.1 твердження (1) - (4) щодо кібернетичного підходу до вирішення загальної проблеми структурної

адаптації УПС в процесі її супроводу, для того, щоб позбавитись негативних наслідків НФ, у процесі супроводу УПС необхідно застосувати нові технологічні рішення та множину метрик оцінки якості отриманих при цьому модифікованих програмних рішень, що дозволить забезпечити зворотний зв'язок і реалізувати контур управління (1) – (6).

В літературних джерелах наводиться велика кількість досліджень, пов'язаних із проблемою виявлення НФ на різних рівнях розробки ПЗ [36, 37], оцінки її якісних характеристик та побудови можливих класифікації окремих видів НФ [38, 39]. Досліджується також можлива взаємодія НФ із не наскрізними функціями (властивостями) ПЗ та її розповсюдження у вихідному коді ПС (наприклад, див. у [40, 41]). Типовими прикладами функціональності, що може проявлятися на рівні вихідного коду різних типів УПС як НФ, відповідно до стандарту документації прикладної розробки MSDN [42] можуть бути такі як:

– запис подій в системний журнал (logging),
– управління транзакціями (transaction management),
– валідація даних (data validation),

та деякі інші.

Дослідження різних проявів НФ встановили [40, 42, 43, 44], що вихідному коду, який містить її, притаманні дві основні властивості:

– розсіювання (scattering) та
– заплутування (tangling).

Вихідний код з НФ є *розсіяним* серед програмних класів УПС тому, що стандартні засоби ООП не надають можливості ізолювати його у певних окремих модулях, наприклад: запис подій в системний журнал (event syslog) повинен бути реалізований для всіх базових функцій бізнес-логіки EGC. Вихідний код НФ *переплутується* із базовим (тобто вільним від НФ) кодом функціональності УПС, тому що розсіювання НФ, в свою чергу, вимагає його «вбудови» в різні програмні модулі.

За характером бізнес-логіки, яка може бути уражена НФ, вона може бути розділена на два типи [45]: гомогенна та гетерогенна. Обидва типи НФ передбачають розширення (доповненням) вихідного коду програмних класів основної бізнес-логіки УПС шляхом додавання до них фрагменту коду, що реалізує відповідну НФ, як це наведено на рис. 1.3.

```
public class Rectangle {
    private Point point;
    public void setPointPos(
                        Point point){
          this. point = point;

       Canvas.redraw(this);          }
}
public class Triangle {
    private Point point2;
    public void setPosition(
                       Point point2){
       this.point2 = point 2;

       Canvas.redraw(this);          }
}
```

(а)

```
public class CreditCardOperator{
     Logger log;

   public void debitOperation(Card card,
                                    Cash qt){
         log.write("Starting debiting"
                        + "Card: " + card
                        + " Quantity: " + qt);
// operation progress

          log.write ("Debit operation
                                        accomplished"
                     + "Card: " + card);
     }
//other operation methods
}
```

(б)

Рисунок 1.3 – Фрагмент вихідного коду для гомогенної (а) та гетерогенної (б) НФ

Гомогенна НФ представляє собою однаковий фрагмент вихідного коду, що багаторазово додається в різні частини класів основної логіки УПС. На рис. 1.3, (а) наведений фрагмент вихідного Java-коду двох класів основної логіки, що представляють собою геометричні фігури: *Rectangle* (квадрат) та *Triangle* (трикутник). Вони мають поле (field) типу Point (точка) і відповідні методи (method): setPointPos(…), setPosition(…), для встановлення значень цих полів. У методах присутній виклик метода перерисовування фігури на екрані класу *Canvas* (канва) – це метод *Canvas.redraw(this)*. Цей фрагмент програмного коду

представляє собою гомогенну НФ (що зумовлена вимогою користувача) – оновлення фігури на екрані кожного разу, як до неї буде додана нова точка.

Гетерогенна НФ представляє собою різні за реалізацією фрагменти програмного коду, для різних класів основної логіки, але які логічно належать до реалізації однієї і тієї ж функціональної вимоги користувача. На рис. 1.3 (б) наведений фрагмент вихідного Java-коду класу основної логіки, що представляє собою операції з кредитною карткою: це клас *CreditCardOperator*. Метод цього класу, що відповідає за дебетову операцію *debitOperation (...)*, містить в собі виклик метода для запису в журнал операцій класу *Logger*: це метод log.*write* (…). Функція запису в журнал виконується на початку та після завершення кожної операції та представляє собою гетерогенну НФ. Гетерогенність цієї НФ зумовлена двома чинниками: 1) повідомлення про початок операції відрізняється від повідомлення про її завершення; 2) повідомлення для інших операцій також буде відрізнятися її назвою та значеннями її параметрів.

Наслідком присутності в УПС коду, що реалізує гомогенну або гетерогенну НФ, є наступні проблеми у процесі її супроводу [46], а саме:

- ускладнюється відстежуваність артефактів розробки, наприклад відстежуваність вимог [47];
- неможливо повторно використати програмний код, що реалізує НФ, через його низьку модульність;
- вихідний код УПС стає надлишковим (code redundancy);
- знижується розуміння розробниками вихідного коду та функціональності бізнес-логіки, яку він реалізує.

Таким чином, НФ стає однією з суттєвих причин дефектів, що виникають в процесі супроводу УПС [48]. У стандарті «Стандарт з класифікації аномалій для програмних систем» IEEE 1044-2009 (“IEEE Standard Classification for Software Anomalies”) [49] зазначено, що програмний дефект (defect) – це будь-яка невідповідність в функціях ПЗ, з причин якої воно не відповідає вимогам

користувачів. Згідно з наведеної у стандарті класифікації поняття «помилка» (fail) є підтипом поняття «дефект», що призводить до появи певних проблем (problem) при роботі з УПС. В роботі [50] наведені результати дослідження з приводу існування зв'язку (кореляції) між НФ та дефектами ПЗ різних типів. Результати цього дослідження показують наявність певної позитивної кореляції між НФ та наявністю дефектів в ПЗ.

Для усунення негативних наслідків НФ в літературі пропонується відповідний концептуальний підхід при розробці ПЗ: це так зване *розділення інтересів* (separation of concerns – SoC) [51]. Основна ідея цього підходу – це проведення процедури декомпозиції фрагментів вихідного коду, який містить НФ, а потім – виконання процедури композиції «вилучених» фрагментів коду НФ з програмним кодом, що реалізує базову функціональність УПС. Запровадження підходу SoC дозволяє зменшити ступінь зв'язності класів у програмному коді УПС, знизити його надлишковість та повторно використовувати нові програмні модулі, що містять код НФ. Але існуючи в рамках ООП-підходу програмні механізми не забезпечують можливості ефективно реалізовувати процедури розділення інтересів [52] і саме тому після аналізу основних властивостей і негативних проявів впливу присутності НФ на процес супроводу УПС, доцільно розглянути нові підходи до розробки ПЗ, що реалізують принцип SoC та надають відповідний модельно-технологічний інструментарій для вирішення цих проблем при супроводі УПС.

1.3. Причини виникнення та стисла характеристика основних пост об'єктно-орієнтованих технологій (ПООТ) розробки програмного забезпечення (ПЗ)

За останні два десятиліття з'явилися нові підходи до розробки та супроводу програмних систем, які в літературі отримали назву пост об'єктно-орієнтованих технологій (ПООТ), наприклад див. у [53], а саме: аспектно-орієнтований підхід до розробки ПС – АОП (aspect-oriented software development – AOSD), веб-сайт

офіційного співтовариства [54], функціонально-орієнтований підхід до розробки ПС – ФОП (feature-oriented software development – FOSD), веб-сайт офіційного співтовариства [55], контекстно-орієнтований підхід до розробки ПС – КОП (context-oriented software development – COSD), веб-сайт офіційної групи розробників [56]. Впродовж останнього десятиріччя зазначені підходи отримали активний розвиток та є предметом інтересу міжнародних наукових груп і груп розробників, про це свідчать цільові конференції та семінари (workshop), присвячені проблемам розвитку та застосування кожного із зазначених підходів [57, 58, 59] (семінар з КОП проходить в рамках європейської конференції з об'єктно-орієнтованого програмування [60]).

Зазначені підходи базуються на принципах ООП та мають вбудовані механізми, що реалізують концепцію розділення інтересів (SoC [51]). Для подальшого аналізу основних властивостей та механізмів ПООТ під базовою функціональністю (БФ) будемо розуміти методи, що реалізують основну бізнес-логіку УПС і містяться в об'єктно-орієнтованих класах УПС, під НФ-реалізацією (НФР) будемо розуміти функціональність, що була локалізована та видалена із класів БФ, і ізольована в нових ПООТ-модулях.

Аспектно-орієнтований підхід (АОП) розроблений у науково-дослідному центрі компанії Xerox PARC [61] та на сьогоднішній день інструментарій для розробки програмного забезпечення реалізований для багатьох мов програмування, наприклад для Java – AspectJ [62]; для C/C++ – AspectC++ [63]; для C# – PostSharp [64], для JavaScript – AspectJS [65], та ін.

Основною ідеєю АОП є ізоляція вихідного коду, що реалізує НФ у ПООТ-модуль, який називається аспектом (aspect) та подальшою його композицією (weaving) з вихідним кодом БФ у відповідності з точками з'єднання (joinpoint), що визначаються у аспекти (наприклад див. у [66]). Ця ідея була запропонована наприкінці 1990-х років та остаточно базові елементи АОП були визначені та отримали одну з перших реалізацій як мова програмування на початку 2000-х [67].

АОП складається із шести базових елементів:

- аспект (aspect);
- зріз (point cut);
- точка з’єднання (join point);
- повідомлення (advice);
- інтродукція (inter-type declaration);
- компонувальник (weaver);

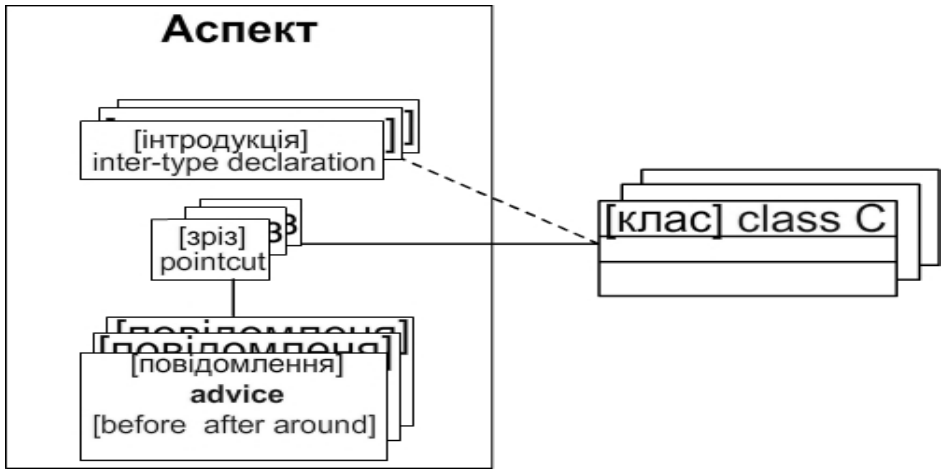


Рисунок 1.4 – Основні конструкції АОП

На рис. 1.4 схематично зображена спрощена структура АОП та зв’язок його елементів між собою, та із зовнішнім класом БФ [68]. Одним із основних елементів АОП є *точка з’єднання*, що дозволяє визначити точки взаємодії аспектна з основними конструкціями ООП (клас, метод, поле), що реалізовані в програмних класах БФ. Точки з’єднання можуть бути визначені для виклику чи виконання методів, або для ініціалізації об’єкту класу, або для доступу до полів класу.

Елементом АОП, що безпосередньо зв’язаний із точкою з’єднання є *зріз*, що конкретизує до яких саме методів, полів, або об’єктів повинен бути приєднаний аспект. Зріз визначається у вихідному коді аспекту.

Повідомлення представляє собою екземпляр фрагменту вихідного коду, копії якого були вилучені із програмних класів БФ і реалізовували логіку НФ. Повідомлення використовують зрізи для подальшої ін’єкції визначеного в ньому фрагменту вихідного коду НФ до всіх точок з’єднання БФ, які описує зріз. За характером ін’єкцій вихідного коду, повідомлення бувають трьох основних типів: апріорна ін’єкція (*before*); апостеріорна ін’єкція (*after*); ін’єкція заміщення (*around*) – логіка повідомлення виконується замість логіки БФ.

За допомогою інтродукції до класів БФ можливо додавати нові елементи ООП (поля, методи), що мають бути визначені в аспекті, та змінювати ієрархію наслідування класу.

Безпосередню композицію вихідного коду повідомлень та інтродукцій, що складають «тіло» аспекту, за правилами з’єднання, що визначені через зрізи та утворені, як правило численні, точки з’єднання, виконує *компонувальник*. В залежності від практичної реалізації АОП композиція може бути активована у два різні моменти, а саме: під час компіляції (compile time), така композиція називається «композиція часу компіляції» (compile-time weaving – CTW), та під час безпосереднього виконання (run time), така композиція називається «композиція часу загрузки» (load-time weaving – LTW).

Так як АОП є розширенням ООП, а його реалізації – надбудовами над конкретними мовами програмування, то під час компіляції лексеми АОП замінюються на лексеми оригінальної мови, таким чином забезпечується виконання аспектів у звичайному для мови середовищі, тобто для виконання аспектів не потрібно встановлювати додаткове середовище виконання.

Таким чином, завдяки *аспектам* (додатковим ПООТ-модулям), що інкапсулюють в повідомленні фрагмент вихідного коду який реалізує логіку НФ, в єдиному екземплярі, *точкам з’єднання* та *зрізам*, що визначають всі місця у вихідному коді БФ, в які потрібно зробити «ін’єкції» фрагменту програмного коду НФ, та *компонувальник*, який виконує композицію, АОП реалізовує механізми

декомпозиції і не інвазійної концепції, визначені концепцією розділення функціональності (Separation of Concerns – SoC). Сучасна версія АОП, деталі його реалізації та використання у поєднанні із іншими сучасними технологіями (для мови Java це інструментарій AspectJ) докладно описані, наприклад, у [69].

Однак, багаторічна практика автора використання АОП у процесі супроводу реальних УПС показує і недоліки цього підходу, а саме:

– важко визначати комплексні зрізи, через одночасне спільне використання для одного повідомлення декількох примітивних зрізів;

– потрібно точно знати які і де будуть утворені точки з’єднання;

– повідомлення, що належать до різних аспектів, тобто, реалізують фрагменти вихідного коду для різних наскрізних функцій (вимог), можуть використовувати один і той же самий зріз. Цей факт викликає конфлікт черги композиції.

Таким чином, потужний на перший погляд підхід АОП, при великому скупченні НФ, що провокує утворенні багатьох нових аспектів, має суттєві недоліки, які повинні бути взяті до уваги.

Функціонально-орієнтований підхід (ФОП) розроблений на основі алгебраїчної моделі GenVoca (Genesis + Voca) , що була запропонована на початку 1990-х років (див., напр., у [70]). У рамках моделі GenVoca програма представлялася як набір базових функцій та додаткових, які шляхом композиції утворювали нову модифікацію ПС. На початку 2000-х років на основі GenVoca було розроблено архітектурну модель для композиції ПС – це технологія AHEAD (Algebraic Hierarchical Equations for Application Design), що представляє кожну функціональність (feature) як ієрархію відповідних артефактів, що входять до неї [70]. На сьогоднішній день ФОП як інструментарій розробки ПЗ реалізовано для різних мов програмування, таких як: для Java це Feature IDE [71] та ATS (AHEAD Tool Suite) [72]; для C++ це FeatureC++ [73], і нарешті, інструментарій для

мовонезалежної композиції функціональностей ПС – це технологія FeatureHouse [74] , що підтримує різноманітні мови програмування.

Основна ідея ФОП базується на принципі покрокового удосконалення (step-wise refinement – SWR), відповідно до якого складна модифікація ПС може бути зібрана із невеликих функціональних модулів (feature modules), що інкрементально додаються один до одного [75]. Відповідно до цього принципу функціональність (feature) - це первинний блок модульності, що інкапсулює вихідний код НФ та їх композиція (composition), шляхом злиття (merge) відповідної програмної реалізації, що у кінцевому рахунку дає можливість отримати нову модифікацію ПС [75].

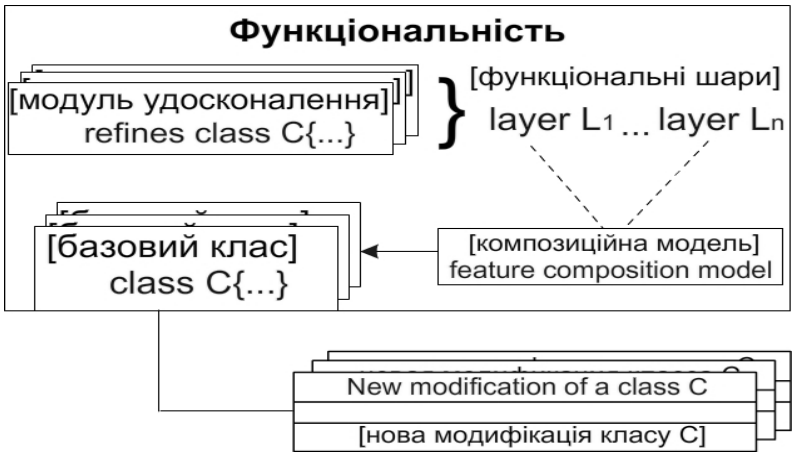


Рисунок 1.5 – Основні конструкції ФОП

На рис. 1.5 схематично зображена спрощена структура ФОП та зв’язок його елементів між собою та із зовнішнім класом БФ [76]. На виході отримується нова модифікація класу, яка складається із функціональності базового класу та нової, доданої, функціональності що містилась у модулі удосконалення.

ФОП складається із базових елементів:

– модуль удосконалення (refine class) –модуль, в якому за принципом SoC повинен міститися програмний код НФ;

– функціональне вдосконалення (refinement) – програмний код НФ, що розширює поведінку (функціональність) базового класу;

– функціональний шар (layer) – визначення функціональності що представляє собою удосконалення. Шари використовуються у моделі композиції;

– композиційна модель (feature composition model) – деревовидна модель, що використовує логічні операції для опису нової модифікації ПС, тобто опису функціональних шарів, функціональне вдосконалення яких повинні бути включені в остаточну модифікацію ПС;

– інтродукція (inter-type declaration) – аналогічна АОП, за допомогою якої, до класів БФ можливо додавати нові елементи ООП (поля, методи);

– композитор (composer) – механізм збірки вихідного коду нової модифікації ПС на базі актуальної композиційної моделі;

Таким чином, за допомогою модулів удосконалення і композитора у ФОП реалізується базові принципи розділення функціональності (Separation of Concerns – SoC): ізоляція фрагмента коду, що належить реалізації НФ у окремих модулях удосконалення, та подальша композиція базової функціональності та функціональності удосконалення в нову модифікацію ПС. Додатковою перевагою такого підходу є чітко визначена модель композиції, тобто усі функції якими повинна володіти конкретна модифікація ПС, яку досить легко змінювати, таким чином включаючи та виключаючи із нової модифікації ПС ті чи інші функції. Іще однією перевагою є те, що ФОП передбачає зв’язок один-до-одного модуля удосконалення та вихідного класу у межах одного функціонального шару, що дає можливість досить легко відслідковувати зв’язки модулів розширення, та ізолювати гетерогенну НФ. При роботі із гетерогенною НФ цей факт є перевагою т.я. набагато простіше відслідковувати який модуль, однак такий підхід має наступний недолік: при роботі с гомогенною НФ має місце дублювання програмного коду, через те що модуль удосконалення в рамках одного

функціонального шару повинен бути визначений для кожного базового класу ООП-коду.

Таким чином, ідея композиції, яку реалізує ФОП, накладає на цей підхід ряд обмежень, що можуть провокувати дублювання коду, зокрема при реалізації функціонального шару, що відноситься до гомогенної НФ.

Контекстно-орієнтований підхід (КОП) розроблений на початку 2000-х років на основі парадигм процедурного програмування (procedural programming), суб'єктного програмування (subjective programming) та на ряду з ними використовує принци об'єктно-орієнтованої парадигми програмування [77]. На сьогоднішній день існує декілька реалізацій КОП для різних ООП-мов програмування [78]. Основна ідея КОП базується на принципі виділення контекстно-залежної поведінки (context-dependent behavior) ПС, що у стандартному ООП реалізується конструкціями *if–else*, що призводить до ускладнення розуміння та супроводу вихідного коду, у функціональні шари (layers), що можуть бути замінені відповідно до контексту в якому використовується програмна система [79].

На рис. 1.6 схематично зображена спрощена структура КОП та зв'язок його елементів між собою та із зовнішнім класом БФ [80]. На виході отримується нова модифікація класу, яка складається із функціональності базового класу та нової, доданої, функціональності що містилась у модулі розшарування, та активується у визначеній контекстом послідовності.

КОП складається із базових елементів, таких як:

– модуль розшарування (layer) – функціональний модуль, що містить в собі вхідний код НФ;

– скупчення шарів (layers) – визначення всіх шарів, що відповідають за контекстно-залежну поведінку (context-dependent behavior) та повинні бути присутні в ПС;

– контекст (behavioral context) – визначення правил активації функціональних шарів.

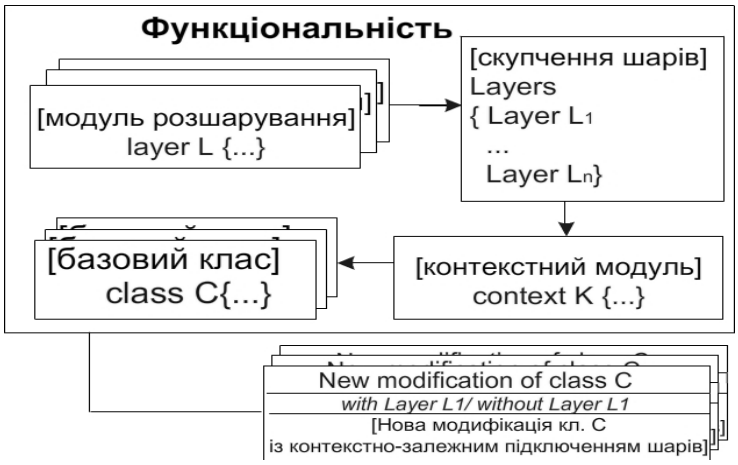


Рисунок 1.6 – Основні конструкції КОП

Таким чином, завдяки визначенню контексту поведінки ПС, відбувається динамічне зв’язування (активація) функціональних шарів (layers), із базовою функціональністю ПС, залежно від умов в яких вона використовується. Недоліком цього підходу є слабка підтримка роботи із гомогенною НФ, що призводить до дублювання вихідного коду.

Для спрощення конструктивні елементи усіх технологій на рис. 1.4 – 1.6, що містять вихідний код функціональності НФ, позначені як функціональні шари; місце фактичного поєднання вихідного коду ізольованої у ПООТ-модулях НФ, та вихідного коду базових класів позначено як точка з’єднання.

Основні якісні властивості досліджуваних ПООТ представлені в Табл. 1.1, де значення експертних оцінок це: «+» коли властивість присутня у відповідній ПООТ; «–» властивість відсутня, «+/–» – властивість присутня частково.

Таблиця 1.1.

Порівняння ПООТ [46]

Властивість	ПООТ		
	АОП	ФОП	КОП
Моделювання поведінки ПС з використанням принципу розділення функціональності (SoC)	+	+	+
Простота збірки нових модифікацій ПС	+	+	+
Підтримка реалізації гомогенної НФ	+	+/–	+
Підтримка реалізації гетерогенної НФ	+/–	+	+
Валив на об’єкти системних класів (вихідний код яких відсутній)	+	–	–
Реалізація функціональних модулів НФ окремо від класу БФ, що підлягає удосконаленню	+	+	+
Простота організації послідовність модифікації однієї точки з’єднання декількома функціональними шарами	+/–	+	+
Динамічна (контекстно-залежна) активація функціональних шарів	–	–	+
Можливість використання декількох підходів одночасно	+/–	+/–	+
Наявність CASE-засобу (поточна реалізації)	+	+	+/–

Отже навіть поверхневий аналіз якісних властивостей досліджуваних ПООТ показує складність їх порівняння між собою та не дає можливості однозначно визначити найбільш ефективну ПООТ для використання в процесі супроводу УПС. Тому для дослідження ефективності їх використання, необхідно розробити додаткові методи та моделі.

1.4. Постановка задачі розробки та дослідження модельно-технологічного інструментарію для оцінки ефективності застосування ПООТ при супроводі УПС

Метою дисертаційної роботи є підвищення ефективності використання пост об’єктно-орієнтованих технологій (ПООТ) у процесі супроводу успадкованих

програмних систем (УПС) шляхом розробки та застосування знання-орієнтованих інформаційних моделей, методів та інструментальних засобів. Для досягнення цієї мети в роботі мають бути вирішені такі задачі:

– формалізувати постановку задачі визначення ефективності застосування пост об’єктно-орієнтованих технологій у процесах супроводу УПС та запропонувати цілісний методологічний підхід до її вирішення;

– розглянути особливості процесу супроводу та структурної модифікації успадкованих програмних систем (УПС), які розроблені на основі ООП, із урахуванням проблем, що виникають при появі ефекту наскрізної функціональності (НФ);

– проведено порівняльній аналіз основних існуючих ПООТ, а саме: аспектно-орієнтованого підходу (АОП), функціонально-орієнтованого підходу (ФОП) та контекстно-орієнтованого підходу (КОП), з метою вивчення особливостей структурних механізмів ізоляції та усунення негативних проявів НФ у УПС;

– розробити загальну модель процесу оцінки ефективності застосування ПООТ;

– запропонувати моделі та засоби для комплексної оцінки стану УПС в процесі їх супроводу;

– розробити архітектурні моделі окремих ПООТ для визначення питомих витрат при їх застосуванні для структурної модифікації УПС;

– запропонувати кількісні метрики для оцінки загального рівня присутності та ступеня розповсюдження НФ у вихідному коді УПС;

– розробити підхід до визначення кінцевої ефективності застосування окремих ПООТ при супроводі УПС із урахуванням багатомірного характеру діючих чинників та критеріїв їх оцінки;

– реалізувати основні програмні компоненти прикладної інформаційної технології для практичного застосування запропонованих моделей та інструментальних засобів;

– провести експериментальне дослідження розробленого підходу та на основі аналізу отриманих результатів розробити практичні рекомендації щодо підвищення ефективності використання ПООТ в процесах супроводу УПС.

Вирішенню цих задач присвячені наступні розділи цієї дисертаційної роботи.

Висновки по розділу 1

У даному розділі дисертаційної роботи:

1) розглянуті деякі критичні проблеми етапу супроводу програмних систем (ПС), що розроблені на основі застосування об’єктно-орієнтованої парадигми (ООП);

2) мотивована актуальність вирішення проблеми усунення негативних наслідків появи наскрізної функціональності (НФ) при супроводі успадкованих ПС (УПС);

3) проаналізовано причини виникнення та надана стисла характеристика основних пост об’єктно-орієнтованих технологій (ПООТ) розробки програмного забезпечення, які мають механізми для усунення проблеми НФ та проведено порівняльний аналіз їх основних якісних ознак;

4) на основі результатів проведеного аналізу сформульовані мета та задачі дисертаційного дослідження.

Нові наукові результати, викладені в цьому розділі, опубліковані в роботах [25, 46, 68, 76, 80].

РОЗДІЛ 2.
МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ МОДЕЛЬНО-ТЕХНОЛОГІЧНОГО ІНСТРУМЕНТАРІЮ ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ПОСТ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ

Цей розділ присвячено розробці методологічних основ побудови моделей та інструментальних технологічних засобів для оцінки ефективності використання ПООТ в процесах супроводу УПС.

У першому підрозділі наведено формалізацію задачі визначення ефективності застосування ПООТ, а також запропоновано загальні методологічні аспекти для подальшого аналізу та моделювання процесів, що впливають на вибір найбільш ефективної ПООТ.

Другий підрозділ присвячено аналізу існуючих підходів до визначення показників структурної складності УПС та мотивації вибору відповідних методів та метрик із урахування особливостей загальної задачі цього дослідження.

У третьому підрозділі розглянуто методи аналізу та оцінки вимог до ПЗ, та запропоновано логіко-лінгвістичний підхід до моделювання та аналізу загального стану вимог, які мають бути імплементовані у процесі супроводу певної УПС.

Четвертий підрозділ присвячено розробці архітектурних моделей для аналізу технологічних особливостей застосування окремих ПООТ з метою подальшої оцінки витрат на їх застосування при модифікації УПС для усунення негативних проявів наскрізної функціональності в її вихідному коді.

П'ятий підрозділ містить аналіз підходів до оцінки характеристик проявів НФ у вихідного коду УПС і результати побудови відповідних метрик для оцінки як рівня присутності, так і характеру розповсюдження НФ у відповідній УПС.

2.1. Формалізація задачі визначення ефективності застосування пост об'єктно-орієнтованих технологій у процесах супроводу УПС і знання-орієнтований підхід до її вирішення

Для комплексного та ефективного опрацювання окремих задач дисертаційного дослідження, які були сформульовані у п. 1.4 попереднього розділу, необхідно провести формалізацію загальної постановки задачі визначення ефективності застосування ПООТ у процесах супроводу УПС та запропонувати певний методологічний підхід для її вирішення.

З метою формалізації основної задачі дослідження пропонується ввести наступні твердження та визначення:

Твердження 1. Існує позитивна кореляція між наявністю наскрізної функціональності в УПС та загальною кількістю дефектів, шо виникають в програмному коді системи в процесі її супроводу (див. п. 1.2 попереднього розділу).

Твердження 2. Зниження рівня НФ можливе шляхом застосування однієї із існуючих ПООТ, що пов'язано із додатковими витратами на реалізацію нових програмних механізмів в УПС (див. п. 1.3 попереднього розділу).

Твердження 3. Кожна з УПС може бути віднесена до певного типу таких систем, тобто існує множина типів УПС, позначена як S

$$(SysType)_i \in S, i = 1,2,3,..., \tag{2.1}$$

де $(SysType)_i$ є позначення для і-го типу УПС.

Твердження 4. Існує множина пост об'єктно-орієнтованих технологій розробки ПЗ, позначена як множина *T*, тобто

$$(POOT)_j \in T, j = 1,2,3,..., \tag{2.2}$$

де $(POOT)_j$ є позначення для j-го типу ПООТ (тобто це наразі вже існуючи технології як АОП, ФОП та КОП – див. п.п. 1.3 попереднього розділу, – та можливі майбутні такі підходи).

Тоді, якщо у разі застосування до УПС i-го типу певної ПООТ j-го позначити можливий рівень зниження НФ як ΔCFR (CFR – Crosscutting Functionality Ratio) та затрати на реалізацію ПООТ як SIC (SIC – Software Implementation Costs), і взяти до уваги вирази (2.1) – (2.2), то у формалізованому вигляді їх можна представити двома наступними виразами:

$$\Delta(CFR)_{i,j} = \delta((SysType)_i, (POOT)_j) \tag{2.3}$$

$$SIC_j = \lambda((POOT)_j), \tag{2.4}$$

де $\delta(...)$ та $\lambda(...)$ – деякі функціональні залежності між відповідними змінними, які визначаються виразами (2.1) – (2.2).

Твердження 5. Коефіцієнт ефективності використання ПООТ j-го типу при супроводі УПС i-го типу може бути визначений як відношення величини можливого зниження рівня НФ у ПС до тих затрат, які пов'язані з застосуванням обраної ПООТ. Враховуючи вирази (2.3) – (2.4), цей коефіцієнт ефективності використання ПООТ можна представити наступним чином:

$$K_{Eff}((SysType)_i, (POOT)_j) = \rho\left(\frac{\Delta(CFR)_{i,j}}{SIC_j}\right), \tag{2.5}$$

де $\rho(...)$ – відповідна функціональна залежність між рівнем зниження НФ із використанням ПООТ та обсягом витрат на її реалізацію в УПС.

Зважаючи на загальновизнану складність отримання кількісних параметрів опису стану складних УПС [4], а також на те, що визначення результатів

застосування існуючих ПООТ вимагає врахування значної кількості слабоформалізованих чинників [46], можна мотивовано стверджувати, що функціональні залежності $\delta(...)$, $\lambda(...)$ та $\rho(...)$ у виразах (2.3), (2.4), які є необхідними для отримання значень коефіцієнта ефективності застосування ПООТ за виразом (2.5), не можуть бути отриманими в аналітичному вигляді.

Враховуючи цей висновок, в подальшому для створення відповідного модельно-технологічного інструментарію для оцінки ефективності застосування ПООТ в процесах супроводу УПС доцільно використати методологічний підхід, який передбачає розробку знання-орієнтованих моделей і методів для структурування та обробки гетерогенних інформаційних ресурсів у відповідній предметній області, який запропоновано в роботах академіка НАН України О.В. Палагіна та його колег з Інституту кібернетики НАНУ [81, 82]. При цьому під знаннями предметної області (domain knowledge) у рамках цього дослідження мається на увазі одне з досить загальних сучасних визначень цього поняття [83], а саме: це структуровані певним чином інформаційні об’єкти, взаємозв’язки між ними та відповідних семантичних правил для їх обробки (інтерпретації), які дозволяють встановлювати нові достовірні факти щодо процесів (подій), що відбуваються у певній предметній області. Слід зазначити, що саме такий знання-орієнтований підхід до розробки інтелектуальних інформаційних технологій вже досить успішно застосовувався для вирішення таких задач як, наприклад, трасування вимог в гнучких процесах розробки програмних систем [84] і підвищення ефективності функціонування систем управління IT-інфраструктурою організацій [85].

Беручи до уваги вирази (2.1) – (2.5), концептуальну схему вирішення задачі визначення ефективності застосування ПООТ для модифікації відповідної УПС в процесі її супроводу можна представити за допомогою метафори багатовимірного інформаційного простору (БІП), побудова якого розглянута в [46]. Графічна інтерпретація метафори БІП представлена на рис. 2.1, де кожна вісь представляє

собою одну з трьох комплексних інформаційних характеристик, які є необхідними для кінцевої оцінки ефективності застосування ПООТ, а саме:

- вісь абсцис відображає окремі типи УПС: I, II, III, IV …. з їх множини S **,** яка визначена формулою (2.1));
- вісь ординат містить показники питомих витрат на модифікацію УПС із застосуванням відповідної ПООТ із множини $\boldsymbol{T}$ - див. вираз (2.2);
- і, нарешті, вісь аплікат репрезентує значення коефіцієнту ефективності використання відповідної ПООТ для УПС заданого типу - це є коефіцієнт K_{Eff}, (див. вираз (2.5)).

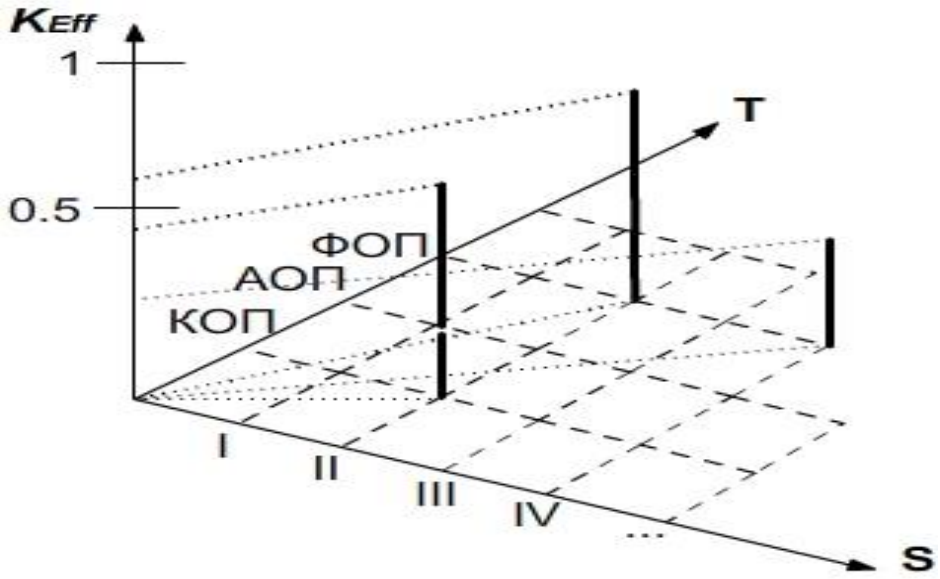


Рисунок 2.1 – Метафора багатовимірного інформаційного простору для моделювання ефективності використання ПООТ при супроводі УПС

Використовуючи метафору БІП і запропонований знання-орієнтований методологічний підхід, в подальшому пропонується розробити сукупність моделей, експертних методів та інструментальних засобів, які дозволять отримати відповідні кількісні оцінки показників, що входять до виразу (2.5), шляхом послідовного вирішення наступних взаємопов’язаних комплексних задач [86]:

Задача 1. Розробити підхід до визначення типу УПС із урахуванням як її структурної складності так і стану функціональних вимог, які мають бути реалізовані в процесі її супроводу.

Задача 2. Розробити метод для обчислення витрат на програмну реалізацію модифікованої архітектури УПС із використанням певної ПООТ.

Задача 3. Побудувати метрики для оцінки рівня присутності НФ у програмному коді УПС, див. вираз (2.3);

Задача 4. Запропонувати підхід до кількісного визначення кінцевої ефективності застосування відповідної ПООТ до заданого типу УПС із урахуванням результатів вирішення задач 1 – 3.

В подальшому у цьому розділі розглянуто підходи до розробки моделей та інструментальних засобів, які мають забезпечити ефективне вирішення *Задач 1-3* на основі запропонованого знання-орієнтованого підходу. Вирішення *Задачі 4,* яке має інтегрувати в собі результати попередніх етапів, детально представлено в наступному, третьому розділі дисертації.

2.2. Аналіз і мотивований вибір експертних методів та кількісних метрик для оцінки структурної складності УПС

Для вирішення *Задачі 1,* як це передбачає запропонований в попередньому підрозділі знання-орієнтований підхід, необхідно мати можливість отримувати оцінку структурної складності та визначати стан вимог до УПС.

Слід зазначити, що на теперішній час існує досить багато альтернативних підходів до визначення складності ПЗ із застосуванням різних показників та методик їх обчислення (див. наприклад у [87, 88, 89, 90, 91]). Враховуючи те, що в цьому дослідженні аналізуються саме технологічні аспекти процесів розробки та супроводу ПЗ, в подальшому в ньому не будуть розглядатися підходи, що орієнтовані на оцінку економічних показників процесів розробки ПЗ, а також такі,

що є орієнтованими на оцінку таких показників процесу створення ПС, як наприклад, чисельність команди розробників систем, їх кваліфікація та досвід і таке інше (див., напр. в [92]).

В свою чергу, підходи до оцінки структурної складності ПС, які враховують саме архітектурні та технологічні аспекти її побудови, можна умовно розділити на дві великі групи, а саме:

(1) підходи, що застосовуються для оцінки функціональної складової ПС, та

(2) підходи, що фокусуються на оцінці складності структур даних та програмного коду системи.

До підходів першої групи відносяться підходи, які використовують для процесу оцінки специфікації вимог проектні документи (схеми, діаграми тощо), а вже потім корегування отриманих таким чином оцінок відбувається із залученням вихідного програмного коду ПС. Найбільш поширеними серед таких методів є метод аналізу функціональних балів (function points analysis) та метод аналізу об’єктних балів (object points analysis).

Метод функціональних балів на сьогоднішній день є досить розвиненим, про що свідчить велика кількість відповідних ISO-стандартів (див. наприклад у [93, 94, 95], і він ввикористовується, наприклад, у відомій моделі якості ПЗ – моделі COCOMO II (Constructive Cost Model) [96]. Зазначений метод дозволяє оцінювати складність ПС в функціональних балах, які являють собою одиниці складності ПС, що є незалежними від конкретної мови програмування та від середовища розробки ПЗ. Метод передбачає декомпозицію ПС на п’ять складових, розділених на дві категорії: функціональні елементи даних (data functions) та функціональні елементи обміну даними (transaction functions) [95]. Зазначений метод передбачає також можливість отримати оцінку складності ПС у строках вихідного коду (Source Line Of Code – SLOC), шляхом застосування спеціальної процедури переводу отриманих функціональних балів в SLOC. Зважаючи на указані факти метод функціональних балів є досить універсальним.

Метод об'єктних балів теж є досить розвиненим (див., напр., у [97]) та має різні модифікації. Він також використовується в моделі COCOMO II на для оцінки складності ПС на стадії її можливого прототипування (prototyping) [96]. Характерною рисою цього метода? на відміну від методу функціональних балid, є використання більш складних об’єктів, основними із яких є наступні: екран (screen), звіт (report) та інтерфейс (interface). Об’єкти екрану (або відображення GUI) та звіту відображають або певним чином інтерпретують дані в ПС, а об’єкт інтерфейсу відповідає за зв’язок ПС із зовнішніми системами (модулями). На відміну від методу функціональних балів, цей метод є простішим у його вивченні та подальшому використанні, а також має деякі інші переваги [98].

До підходів другої групи можна віднести такі, що залучають до оцінки складності ПС метрики вихідного коду, порівняльний аналіз яких представлено в роботах [99, 100]. На підставі цього усі такі метрики можуть бути поділені на певні категорії, у відповідності з тим, яку саме характеристику ПС вони аналізують. Зокрема, до них належать такі, як: складність (complexity), інкапсуляція (encapsulation), наслідування (inheritance), зв’язність (coupling), зчеплення (cohesion), абстрактність (abstractness), та ін.

Слід зазначити, що у разі необхідності модифікації вже існуючої УПС, досить часто єдиним інформаційним ресурсом для команди розробників, які займаються її супроводом, є саме вихідний програмний код системи [6]. Саме тому в подальшому перевага віддається застосуванню метрик оцінки складності вихідного коду які наведені в Табл. 2.1. Крім того, ці метрики не залежать від мови програмування, яка була використана при реалізації УПС, на відміну від таких метрик, як наприклад кількість строчок коду (Line Of Code – LOC) [101], що, в свою чергу, є суб’єктивним показником, який досить часто залежить від специфіки конкретної мови програмування та від кваліфікації розробника ПЗ.

Із урахуванням цих міркувань, для формування подальшого методологічного підходу до визначення структурної складності УПС

пропонується розглядати такі метрики, що дозволяють визначати її на трьох логічних рівнях побудови ПЗ, а саме

1. Рівень структурної складності окремого класу (class) – це базовий внутрішній рівень організації програмного коду і тому множина метрик, що використовуються на цьому рівні, оперують такими структурними елементами класу як: поле, метод; та зв’язки між ними, (наприклад див. у [102]). У якості таких метрик пропонується використовувати наступні:

а) цикломатична складність Мак Кейба (McKeyb’s Cyclomatic Complexity) V(G);

б) зважена насиченість класу (Weighted Methods per Class) WMC;

в) недостатність зв’язності методів (Lack Cohesion of Methods)LOCOM.

2. Рівень взаємодії між класами, які належать до певного функціонального пакету (package) – це проміжний рівень структурної організації програмного коду і відповідна множина метрик цього рівня повинна оцінювати зв’язок між різними класами УПС, (наприклад див. у [99, 102]). До таких метрик відносяться:

а) зчеплення між об’єктами (Coupling Between Objects) CBO;

б) глибина дерева наслідування (Depth of Inheritance Tree) DIT;

3. Рівень взаємодії між окремими пакетами – це зовнішній рівень структурної організації програмного коду, множина метрик для визначення його складності повинна оцінювати зв’язки між окремими функціональними пакетами, які містять в собі відповідні множини програмних класів. У якості таких метрик пропонується використовувати наступні [103]:

а) нестабільність пакета (Instability) I(p);

б) абстрактність пакета (Abstractness) A(p);

в) відстань від головної послідовності (Distance from the Main Sequence) D(p);

Більш детально запропоновані метрики визначаються у наступний спосіб.

(1.а) Цикломатична складність МакКейба V(G) застосовується для визначення складності метода. Для розрахунків використовується орієнтований сильно-зв'язний граф. Цикломатичне число МакКейба показує необхідну кількість проходів для покриття усіх контурів сильно-зв'язного графа, або кількість можливих шляхів виконання метода, які зумовлені наявністю в ньому виразів умови.

$$V(G) = E - N + 2P, \tag{2.6}$$

де E – кількість ребер орієнтованого графа,

N – кількість вузлів на графі,

P – число з'єднаних компонентів графа.

В процесі автоматизованого обчислення показника цикломатичної складності, як правило, застосовується спрощений підхід, у відповідності з яким не виконується побудова графа, натомість обчислення зазначеного показника проводиться на основі підрахунку кількості виразів умови (наприклад, таких, як if ... then … else, switch of …. та ін.) і можливої кількості альтернативних сценаріїв виконання метода.

(1.б) Зважена насиченість класу (WMC) накопичує цикломатичну складність у класі K з методами $M_1 ... M_n$, які визначені в цьому класі але не є успадкованими від предків.

$$WMC = \sum_{i=1}^{n} C_i, \tag{2.7}$$

де C – цикломатична складність методів класу.

(1.в) Недостатність зв'язності методів (LOCOM) накопичує кількість нульових пар методів, які не мають спільних атрибутів

$$LOCOM = \frac{m - \left(\frac{1}{a} \sum_{j=1}^{a} \mu(A_j) \right)}{m - 1}, \tag{2.8}$$

де m – кількість методів класу,

a – кількість атрибутів класу,

$\mu(A)$ – атрибут, що використовується в методі.

(2.а) Зчеплення між об’єктами (CBO) оцінює взаємодію між об’єктами класів. Ця метрика фіксує кількість сторонніх класів, з якими зв’язаний досліджуваний клас, окрім успадкованих, класів системних пакетів, та примітивних типів. Ця метрика враховує тільки перше входження стороннього об’єкта в досліджуваний клас. Дуже високий показник зчеплення між об’єктом одного класу та об’єктом іншого класу може обмежувати модульність, що ускладнює тестування та супровід.

$$CBO = \sum_{o=1}^{d} I_o, \tag{2.9}$$

де I – будь-який програмний об’єкт стороннього класу,

d – кількість об’єктів стороннього класу, з якими зв’язаний цей клас.

(2.б) Глибина дерева наслідування (DIT) – це множина класів, які є предками (predecessors) даного класу. Якщо клас не має предків, в цьому випадку значення DIT, дорівнює 1.

$$DIT = \#(Predecessor), \tag{2.10}$$

де *#(Predecessor)* – кількість класів-предків, даного класу.

(3.а) Нестабільність пакета I(p) відображає на скільки пакет «залежний» або «не залежний» від інших пакетів, та на скільки він може піддаватися.

Умови стабільності пакету X: X не залежить від інших пакетів, тобто в ньому не присутні зв'язки з класами із інших пакетів, а є множина пакетів, що залежать від нього. З огляду на це немає зовнішніх факторів, що можуть спричинити модифікацію пакета X, отже такий пакет є *стабільним,* тобто він не залежним від змін в інших пакетах.

Умови нестабільності пакету X: X залежить від множини інших пакетів, тобто класи, що входять до пакету X використовують класи із інших пакетів, тоді модифікації у множині пакетів від яких він залежить, призведуть до змін пакеті X, такий пакет є *не стабільним*.

$$I(p) = \frac{C_e}{C_e + C_a},\ I(p) \in [0;1] \left| \begin{matrix} 0 \text{ - пакет стабільний} \\ 1\text{- пакет не стабільний} \end{matrix} \right. , \qquad (2.11)$$

де C_a, - це так звана доцентрова зв'язність пакету, або це кількість програмних класів, які знаходяться за межами пакета і при цьому залежать від класів, які містяться всередині пакета, або це кількість вхідних залежностей даного пакету,

C_e, – це відцентрова зв'язність, тобто це кількість класів за межами пакета, від яких залежать класи всередині пакета, або це кількість вихідних залежностей пакету.

(3.б) Абстрактність пакета A(p) відображає ступінь присутності в пакеті абстрактних класів та інтерфейсів, тобто таких, в яких повністю або частково відсутня реалізація конкретних методів, і які потребують додаткової їх реалізації через механізм наслідування

$$A(p)=\frac{N_a}{N_c},\ A(p)\in[0;1]\begin{vmatrix}0 \text{ - пакет "визначений"}\\1 \text{ - пакет "абстрактний"}\end{vmatrix}, \qquad (2.12)$$

де N_a, – це кількість абстрактних класів в пакеті,

N_c, – це загальна кількість класів в пакеті.

Термін «Визначений пакет» означає, що в ньому немає абстрактних класів та інтерфейсів, натомість «Абстрактний пакет» означає, що в ньому містяться тільки абстрактні класи та інтерфейси.

(3. в) Відстань від головної послідовності D(p), яка пов’язана із показниками абстрактності та нестабільності.

$$D(p)=\frac{\left|A(p)+I(p)-1\right|}{\sqrt{2}}, \qquad (2.13)$$

де $A(p)$ – абстрактність пакету, див вираз (2.13),

$I(p)$ – нестабільність пакету, див. вираз (2.12).

Нормалізована форма виразу (2.14):

$$D=\left|A(p)+I(p)-1\right|\ \mathrm{D}\in[0;1], \qquad (2.14)$$

де 0 - означає, що пакет знаходиться на основній послідовності,

1 – пакет знаходиться далеко від основної послідовності.

На рис. 2.2 приведена графічна інтерпретація метрики «Відстань від головної послідовності». При цьому термін «Зона недоцільності» свідчить про те, що пакет X (1;1) з такими характеристиками є абстрактним, і класи в інших пакетах не залежать від класів пакету X, тобто наявність такого пакета в програмному коді УПС недоцільна.

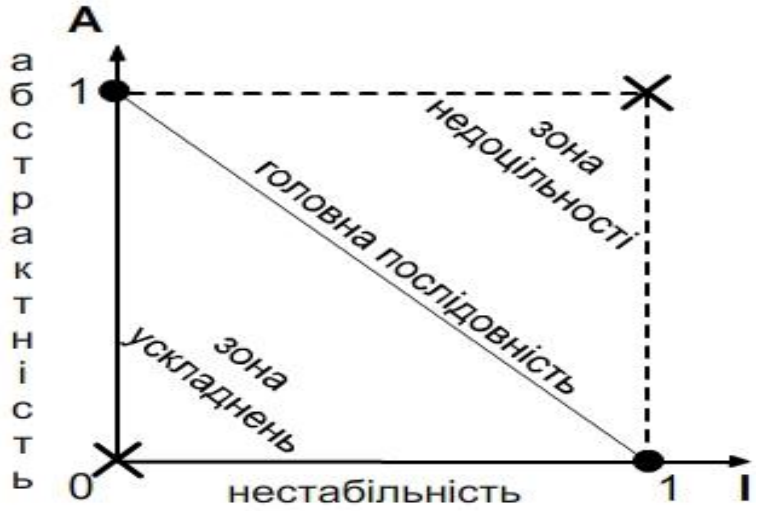


Рисунок 2.2 – Метод головної послідовності [103]

Термін «Зона ускладнень» свідчить про те, що пакет X (0;0) з такими характеристиками не містить абстрактних класів та має багато вхідних залежностей, тому модифікація такого пакету в процесі супроводу УПС буде ускладненою.

Перевага вибору метрик оцінки об’єктно-орієнтованого коду для вимірювання структурної складності УПС полягає в їх різноманітті, що дозволяє обрати таку колекцію метрик, яка найбільш задовольняє потреби конкретного проектного рішення. Однак на ряду з цим, недоліком такого різноманіття є відсутність механізму їх об’єднання у єдиний комплексний показник, тобто порівнювати показники складності декількох програмних систем доводиться попарно, що не дає повного уявлення яка із ПС більш складна.

Разом із тим, використовуючи певну комбінацію метрик, які представлені виразами (2.6) – (2.14) можливо отримати кількісні оцінки структурної складності (Structural Complexity – SC) вихідного коду відповідної УПС, у процесі супроводу якої має бути застосована та чи інша ПООТ:

$$SC = \sum_{i=1}^{8} w_i M_i \,, \tag{2.15}$$

де SC – структурна складність УСП,

M_i – відповідна метрика із колекції метрик об’єктно-орієнтованого вихідного коду, представленої виразами (2.6) – (2.14),

w_i – ваговий коефіцієнт для метрики M_i, при цьому $\sum_{i=1}^{8} w_i = 1$.

Для остаточної оцінки структурної складності УПС необхідно розрахувати відповідні вагові коефіцієнти для зазначеної колекції метрик, що буде розглянуто у наступному розділі цього дослідження.

2.3. Логіко-лінгвістичний підхід до моделювання та аналізу стану вимог до УПС в процесі її супроводу

Вирішення другої частини *Задачі 1* з їх переліку, який був сформульований у п. 2.1 цього розділу, має забезпечити оцінку стану вимог до УПС, що мають бути реалізовані в процесі її супроводу (див. схему на рис. 1.2). В літературних джерелах [104, 105, 106] приводиться декілька визначень терміну «вимога до ПЗ». У стандарті ISO/IEC 24765:2010 [104], зокрема, *вимога* визначається як задокументовані умови або можливості, якими повинна володіти програмна система в цілому або її окремі компоненти, щоб виконати контракт або задовольняти стандартам, специфікаціям або іншим формальним документам.

Відповідно до [104, 105], всі вимоги до ПС розділяються на функціональні та нефункціональні, і вони можуть бути подані у вигляді схеми класифікації, яка представлена на рис. 2.3 у форматі діаграми варіантів використання (Use-Case diagram), нотації UML 2.0. Згідно з нею, *Бізнес-вимоги* (Business Requirements) –це високорівневі цілі, які мають бути досягнуті у відповідній організації завдяки

використанню ПС, що має бути розроблена. Ці вимоги визначають концепцію створення всієї системи, і вони визначаються в спеціальному документі (Vision Document). *Вимоги користувача* (User Requirements) – це безпосередньо множина тих потреб користувачів, які має забезпечити використання цієї ПС, вони також фіксуються в спеціальному проектному документі, який містить опис варіантів використання ПС (Use-Case Document).

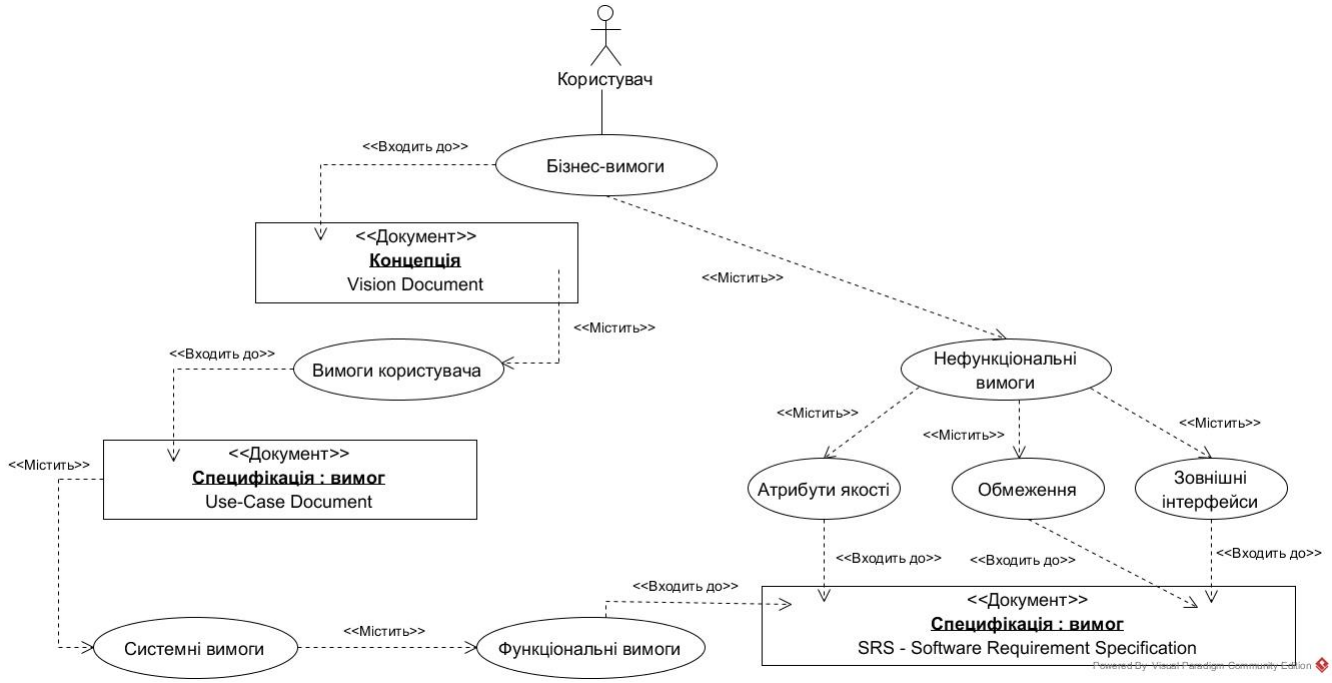


Рисунок 2.3 – Класифікація вимог [105]

Системні вимоги (system requirements) – це високорівневі вимоги до загальних функцій усіх підсистем ПС, які мають бути визначені на підставі аналізу та моделювання вимог користувачів.

Функціональні вимоги (functional requirements), або в подальшому ФВ – це вимоги, що безпосередньо визначають можливості ПС щодо обробки даних та отримання необхідних користувачам результатів.

Нефункціональні вимоги (НФВ) до ПС – це характеристики атрибутів якості (quality attributes), які мають бути забезпечені в процесі її розробки, до них

належать [107]: продуктивність (performance), надійність (reliability), безпечність (safety), переносимість (portability), зручність використання (usability) та супроводжуваність (maintainability). До НФВ належать також так звані *Зовнішні інтерфейси* (external interfaces) – це додаткові вимоги щодо реалізації інтерфейсів взаємодії даної ПС із зовнішніми системами або підсистемами, із урахуванням наявних апаратних пристроїв, каналів зв'язку, форматів передачі даних та ін., а також *Обмеження* (constraints) – це певні додаткові умови, виконання яких має бути враховано при розробці даної ПС. У кінцевому рахунку, усі ФВ та НФВ мають бути зафіксовані у документах з специфікації вимог (software requirements specification – SRS) [108].

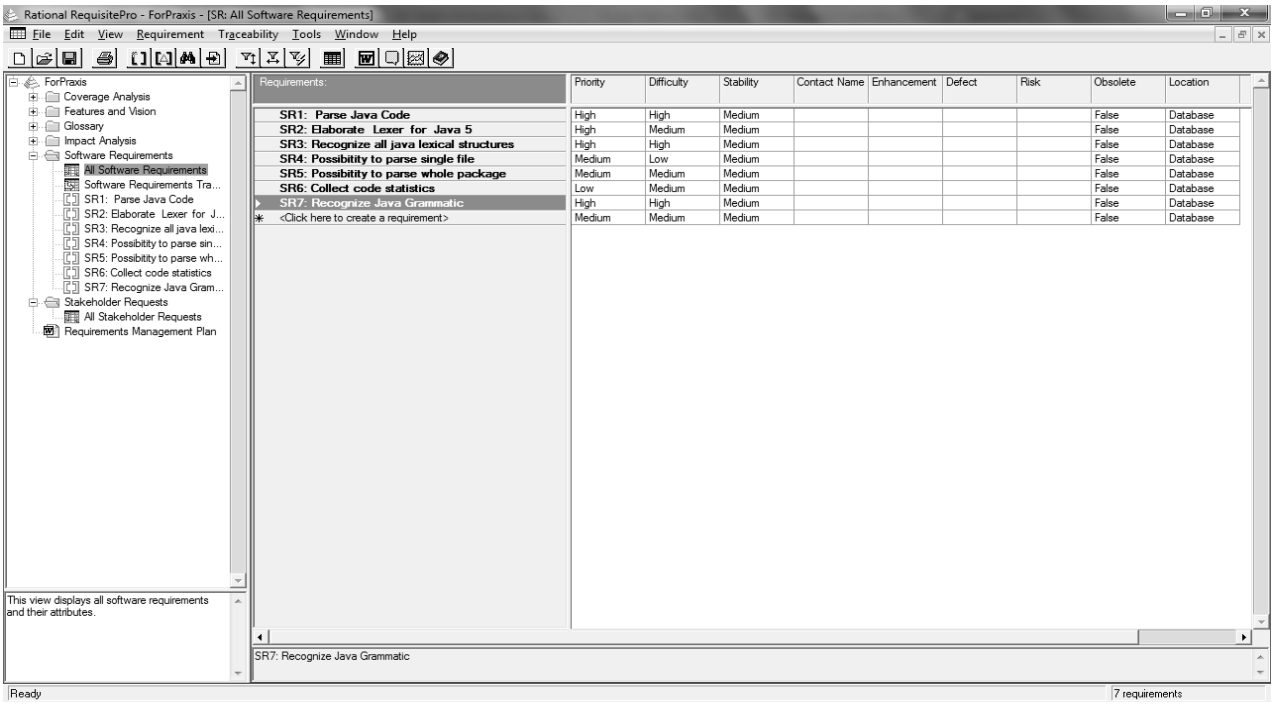


Рисунок 2.4 – Інтерфейс СУВ Rational Requisite Pro

Для автоматизації процесу управління вимогами на сьогоднішній день існує багато комп'ютеризованих систем управління вимогами (СУВ) (requirement management system), які дозволяють проводити первинний текстовий опис вимог, відстежувати їх версії, проводити їх аналіз та їх. До таких систем можна віднести

наступні: IBM Rational Requisite Pro [109], Accompa [110], IBM DOORS Next Generation [111], Gatherspace [112] та деякі ін. Приклад інтерфейсу для роботи аналітика із списком вимог у СУВ IBM Rational Requisite Pro (ця система має академічну ліцензію для некомерційного використання), наведено на рис. 2.4.

Основними атрибутами, які характеризують вимоги в СУВ є такі: дата створення вимоги (date); дата виконання (due by) – дата, коли вимога має бути виконана; джерело (source) – людина, документ, або інше джерело цієї вимоги; пріоритет виконання (priority) – показник важливості виконання вимоги відносно інших; складність вимоги (complexity / difficulty) – оцінка зусиль, що необхідні для реалізації вимоги; статус (status) – показник, що означає, чи прийнята ця вимога до виконання, або вона знаходиться на розгляді та підлягає обговоренню; виконавець вимоги (assigned to) – це розробник, який є відповідальним за реалізацію вимоги; номер версії (version) специфікації вимоги та деякі інші.

Аналіз основних атрибутів вимог, що представлені у більшості сучасних СУВ показує, що с точки зору процесу супроводу УПС, найбільш суттєвими атрибутами, що характеризують вимоги та при цьому вимагають подальшого аналізу та оцінки, є *пріоритет* та *складність* вимоги.

Пріоритет виконання ФВ (надалі *пріоритет ФВ*) – це оцінка відносної важливості виконання цієї вимоги в процесі супроводу УПС [113]. Аналіз деяких СУТ, а також систем управління завданнями (issue tracking system) та систем управління програмними дефектами (bug tracking system) показує [109, 110, 111, 112], що пріоритет ФВ диктується необхідністю швидкої реалізації ФВ та визначається експертним шляхом на основі висновків зацікавлених осіб (stakeholder): представників замовників, менеджерів проекту тощо. У більшості СУТ пріоритет ФВ визначається фіксованою множиною лінгвістичних значень, наприклад: «нейтральний» (neutral), «актуальний» (actual), «негайний» (immediate).

Складність ФВ – це показник оцінки рівня витрат, які є необхідними для реалізації даної вимоги. В свою чергу, в інтерфейсі користувача більшості СУТ (див. приклад екранної форми на рис. 2.4) складність ФВ також визначається лінгвістичною змінною, яка приймає наступні значення: «висока» (high), «середня» (medium), «низька» (low).

Найпоширенішими підходами до кількісної оцінки складності вимог є експертні судження, які використовують при цьому наступні засоби:

- метод балів оцінки варіантів використання (Use Case Points – UCP) [114, 115],
- метод виміру об’єму функціональності (Functional Size Measurement – FSM) [116],
- метод ранніх функціональних балів (Early Function Points – EFP) [117, 118] та деякі інші.

Оцінка складності вимог базується на оцінках експертів у даній предметній області, що мають досвід роботи з аналогічними проектами. Для процедури експертної оцінки можуть використовуватися методи такі як наприклад метод Дельфі (див. напр. у [119]), який спрямовано на вироблення спільної позиції відносно складності ФВ у групи експертів. Такі оцінки можуть бути отримані досить швидко навіть в умовах недостатньої деталізації ФВ, але в той же час вони є вельми приблизними.

Метод балів оцінки варіантів використання направлений на отримання оцінки складності прецедентів та їх сценаріїв, які мають бути побудовані на основі детального опису ФВ. Цей метод базується на визначенні типів акторів (actor), які є задіяними в сценаріях, а саме

- простий актор: він представляє собою деяку іншу ПС, яка взаємодіє з даною УПС через чітко визначений прикладний програмний інтерфейс (Application Programming Interface - API);

– актор середньої складності: це також інша ПС, яка взаємодіє з даною УПС через достатньо складні прикладні протоколи (мережевого та прикладного рівня;
– складний актор: це користувач УПС, який взаємодіє із нею через графічний інтерфейс користувача.

Складність самих сценаріїв виконання прецедентів оцінюється аналогічно, шляхом поділу їх на певні категорії за кількістю сутностей даних, що залучені до опрацювання відповідного сценарію. Загальна складність акторів та варіантів використання отримується шляхом підсумування кількості їх сумарних балів за відповідними категоріями (Unadjusted Use Case Points – UUCP). Іншим аспектом розрахунку є розрахунок складності технічних факторів (Technical Complexity Factor – TCF), складності факторів оточення (Environment Complexity Factor – ECF), та фактору продуктивності (Productivity Factor – PF), що інтерпретує кількість людино/годин, які потрібні на виконання одного балу варіантів використання і лежить у діапазоні від 15 до 30 люд./год. Таким чином, метод балів варіантів використання дозволяє більш точно оцінити складність ФВ у кількісному вигляді, але самі по собі бали варіантів використання не є інформативними, а вибір конкретного значення для переводу їх у людино/години також вимагає досвіду роботи з іншими подібними УПС, та врахування кваліфікації розробників, що входять до команди супроводу УПС.

Стандарт ISO/IEC 14143.1:2007 регламентує концепцію вимірювання функціональної складності ПС на базі оцінки ФВ, та основні принципи використання методів функціонального вимірювання (наприклад, див. в [93, 94, 95]). Основними поняттями цього стандарту є наступні:

– функціональне вимірювання (functional size measurement): це процес вимірювання функціонального розміру;
– функціональний розмір (functional size): це розмір ПС, отриманий шляхом підрахунку кількості ФВ користувача;

– функціональні вимоги користувача (functional user requirements): це множина базових функціональних компонентів ФВ, які повинні бути реалізовані в ПС для задоволення потреб користувача;

– базові функціональні компоненти (base functional component) – елементарна одиниця ФВ користувача, визначена для та яка використовується в методах функціонального вимірювання; та ін.

Зазначений стандарт також характеризує методи функціонального вимірювання, як такі, що повинні задовольняти наступні вимоги:

а) вони можуть бути застосовані до ФВ користувача як тільки вимоги чітко визначені, та задокументовані;

б) вони повинні визначати функціональну складність вимог на основі оцінки базових функціональних компонентів, та правила їх ідентифікації у ФВ користувача;

в) вони повинні коректно визначати та типізувати базові функціональні компоненти;

г) вони повинні визначати правила оцінки та одиниці вимірювання базових функціональних компонентів.

Загальна схема функціонального вимірювання складності вимог приведена на рис. 2.5 в нотації IDEF0 [120].

Процес вимірювання складається із трьох основних блоків активностей: A1, A2 та A3, які зображені відповідними прямокутниками та представляють собою наступні:

– A1: «Вибір методу оцінки функціональної складності ФВ користувача»;

– A2: «Ідентифікацію базових функціональних компонентів»;

– A3: «Вимірювання складності ФВ».

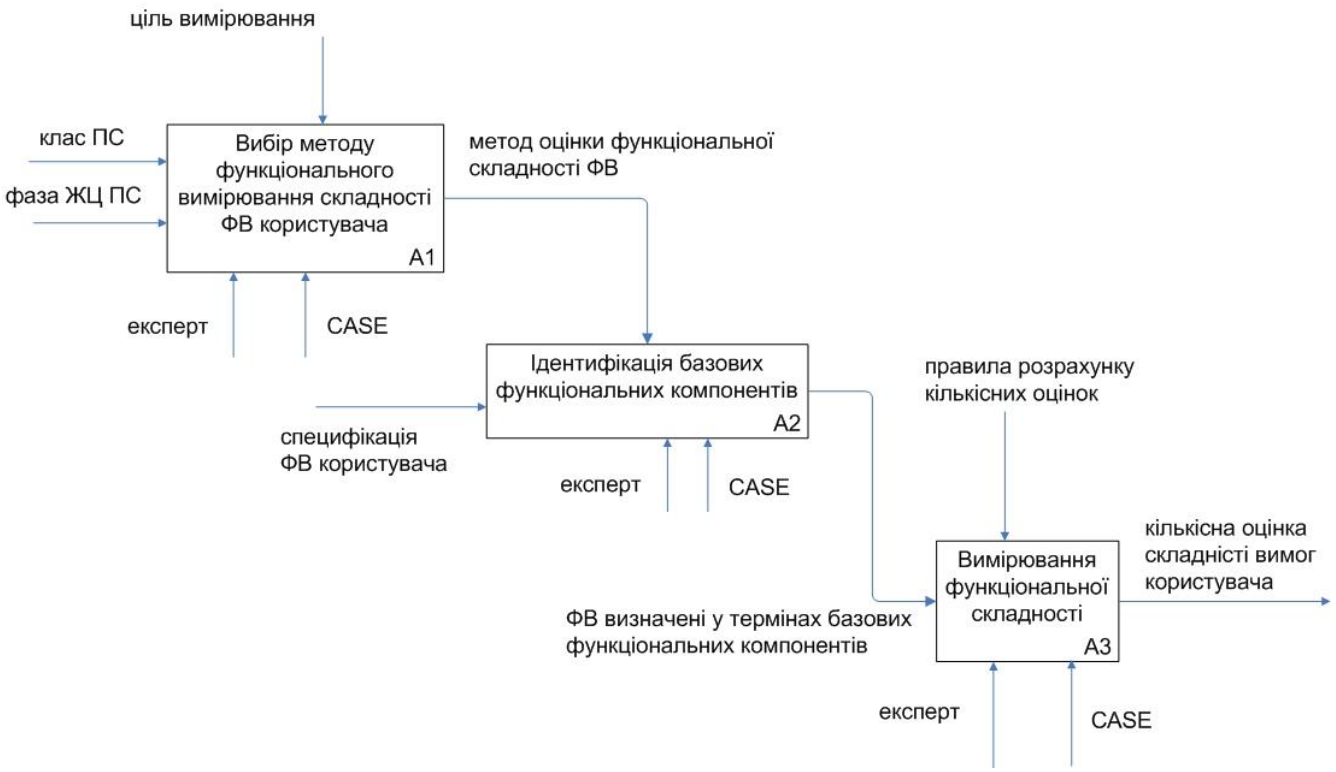


Рисунок 2.5 – Загальна схема процесу вимірювання функціональної складності вимог до ПЗ

У відповідності до нотації IDEF0 кожен блок має наступні інтерфейси: інтерфейс «Вхід» (стрілка, що входить у блок зліва), інтерфейс «Вихід» (стрілка, що виходить із блоку справа), інтерфейс «Управління» (стрілка, що входить в блок зверху), та інтерфейс «Механізм» (стрілка, що входить у блок знизу). Таким чином, для блоку А1 необхідно мати дані про фазу життєвого циклу ПС (в контексті даного дослідження це – фаза супроводу ПЗ) та дані про клас УПС, тобто сфера її застосування (наприклад, інформаційно – аналітичні системи, системи реального часу, мобільні системи тощо). Ціль вимірювання для блоку А1 є інтерфейсом управління, а результатом – це обраний експертами один із методів функціонального вимірювання, який у свою чергу управлінням для блоку А2. Входом для цього блоку є специфікація конкретних ФВ до системи, а результатом його виконання є перелік базових функціональних компонентів, що є необхідними для реалізації цих ФВ. І, нарешті, блок А3 безпосередньо обчислює оцінку функціональної складності вимог за правилами розрахунку попередньо обраного

методу (див. блок А1). Слід зазначити, що схема, запропонована на рис. 2.5, відповідає підходу до визначення методу виміру функціональної складності ФВ за стандартом консорціуму COSMIC [94]).

Метод ранніх функціональних балів (early function point analysis – EFPA) дозволяє оцінити складність вимог на різних рівнях їх деталізації («загальна деталізація», «середня деталізація», «детальна специфікація»), що дає можливість представити кожну ФВ у термінах функціональних компонентів, таких як:

1. *Макрофункція* (macro-function – MF): вона представляє собою великі підсистеми корпоративних ПС, або систему в цілому.

2. *Функція* (functions – F): вона представляє собою підсистеми даної системи, що виконують певну підмножину ФВ користувача, такі функції можуть бути агрегованими у макрофункції.

3. *Мікрофункція* (micro-function – mF): це множина базових операцій обробки даних CRUD: Create (створення), Read (читання), Update (оновлення), Delete (видалення).

4. *Функціональний примітив* (functional primitive – fP): це будь-яка атомарна системна функція, що є важливою для користувача ПЗ (наприклад, виклик контекстної допомоги тощо).

5. *Логічна група даних* (logical data group – LDG): це будь-які структуровані певним чином дані, що мають бути опрацьовані в системі.

Згідно із методом EFPA, кожний функціональний компонент може бути агрегований у функціональний компонент більш високого рівня та може бути інтерпретований на одному з трьох рівней складності: простий, середній, високий. Таким чином, рівень загальної деталізації вимог представляє собою ідентифікацію макрофункцій та функцій. Рівень середньої деталізації ФВ представляється ідентифікацією мікрофункцій та функціональних примітивів. Рівень детальної специфікації представляється додатковою ідентифікацією логічних груп даних. У [117], приведена детальна процедура EFPA, та приклад розрахунку складності ФВ

та зазначається, що похибка між розрахунками на загальному рівні деталізації та детальному рівні специфікації становить близько 15-20%. Отже, метод EFPA дозволяє швидко отримати первинну оцінку складності вимог при їх недостатній деталізації, використовуючи при цьому базові функціональні компоненти високого рівня, та згодом, при отриманні додаткової інформації відносно ФВ, уточнювати оцінку, залучаючи для цього функціональні компоненти нижчого рівня.

Як вже було зазначено вище, у цьому дослідженні основними характеристиками ФВ, що є важливими для аналізу стану УПС у процесі її супроводу, є *пріоритет* (priority) та *складність* (complexity) ФВ. Вирішення *Задачі 2* (див. підрозділ 2.1) передбачає визначення початкової характеристики УПС - це тип системи *T*, у формалізованому вигляді його можна представити як кортеж із двох елементів:

$$T = (SC, R), \tag{2.16}$$

де *SC* – показник структурної складності УПС,

R – показник стану ФВ, або ранг вимог (requirement rank).

Формалізовано ранг вимог може бути представлений як кортеж:

$$R = (Priority, Complexity), \tag{2.17}$$

де *R* – ранг вимог ФВ (requirement rank),

Priority – пріоритет виконання функціональних вимог (priority),

Complexity – складність функціональних вимог (complexity).

Як зазначалося вище, пріоритет ФВ є експертною оцінкою, яка носить лінгвістичний характер та позначається у відповідних СУВ лінгвістичною змінною. Відповідно до [121], структура лінгвістичної змінної є наступною:

$$L = \langle X, T(X), U, G, M \rangle, \tag{2.18}$$

де X – назва лінгвістичної змінної,

$T(X)$ – множина термів змінної X, тобто сукупність її лінгвістичних значень,

U – база лінгвістичної змінної, тобто універсальна множина числових значень, з якими асоціюються терми змінної із множини $T(X)$,

G – синтаксичне правило, що породжує терми з множини $T(X)$,

M – семантичне правило, що кожному лінгвістичному значенню змінної X, ставить у відповідність його семантичне значення $M(X)$, тобто визначає вид функції приналежності.

Таким чином, лінгвістична змінна для опису пріоритету вимог, приймає наступний вигляд:

X – «Пріоритет ФВ»;

$T(X)$ – {*нейтральний, актуальний, негайний*},

U – оцінка якісна, для визначення значень функції M використовується кодована шкала {-6, -4, -2, 0, 2, 4, 6} (наприклад див, у [122]),

G – синтаксичне правило, що породжує терми множини $T(X)$,

M – функція приналежності Харінгтона [122]:

$$d = e^{-e^{-R}}, d \in [0,1], \tag{2.19}$$

де d – значення функції Харінгтона,

e – математична константа, основа натурального логарифма,

R – аргумент функції, значення показника із множини U.

Функція Харінгтона має дві точки перетину:

$$d(0) = \frac{1}{e} \approx 0,37 \text{ та } d(0,78) = 1 - \frac{1}{e} \approx 0,63,$$

а також володіє такими властивостями як неперервність та гладкість. Крім того в областях близьких до 0 та 1, її «чутливість» цієї функції є суттєво меншою, ніж у середній зоні її визначення [122].

Для подальшого аналізу стану ФВ їх складність також необхідно виразити у вигляді відповідної лінгвістичної змінної, яка може бути подана у наступний спосіб:

X – «Складність ФВ»;

T(X) – {*низька, середня, висока*},

U – множина функціональних балів, отриманих за допомогою методу EFPA,

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функція приналежності Харінгтона [122].

Таким чином, згідно із виразом (2.17), для отримання кількісної оцінки рангу вимог, необхідно об’єднати показники двох його складових характеристик, вирази (2.18), (2.19). Метод визначення оцінки рангу вимог розглянуто в наступному розділі даного дослідження.

2.4. Побудова архітектурних моделей для аналізу технологічних особливостей застосування ПООТ

Задача 2, а саме: розробка методу для обчислення витрат на програмну реалізацію нової УПС із використанням ПООТ, що сформульована у рамках запропонованого в підрозділі 2.1 знання-орієнтованого підходу, передбачає відображення характерних ознак окремих ПООТ у такому вигляді, який дозволить у подальшому провести їх кількісний порівняльний аналіз. Для цього пропонується застосувати архітектурно-центрований підхід та використати сучасні структурно-функціональні нотації, які дають змогу проводити аналіз

ПООТ на більш високому рівні абстракції – на рівні їх архітектур, не зважаючи на особливості їх можливих програмних реалізації [123].

Одними з найбільш відомих структурно-функціональних нотацій, які передбачають опис будь-якої ПС на архітектурному рівні є наступні:

- уніфікована мова моделювання (Unified Modeling Language – UML) [124],
- мова моделювання систем (Systems Modeling Language – SysML) [125],
- сімейство мов опису архітектур (Architecture Description Languages – ADL) [126, 127, 128].

UML – це мова, яка призначена для проектування програмних систем на різних рівнях абстракції: концептуальному, логічному та фізичному. Остання на сьогоднішній день версія мови UML 2.5 налічує 17 діаграм, що дозволяють відображати різноманітні аспекти процесу розробки, це такі діаграми, як наприклад [124]: структурні діаграми, діаграми поведінки, діаграми взаємодії та ін. Однак, постійний ріст кількості та складності UML-діаграм, призводить до певних проблем у процесі як їх методологічного вивчення, так і практичного застосування UML. У цьому сенсі мова моделювання SysML [129] є більш компактною нотацією, яка була побудована на певній підмножині мови UML. У порівнянні з UML, SysML надає додаткові можливості, що дають змогу будувати моделі взаємодії користувача не тільки з програмними компонентами ПС, а також із апаратним устаткуванням, і враховувати параметри та обмеження, які вони накладають на інші системні елементи [129, 130].

В свою чергу, сімейство мов опису архітектур (ADL) дозволяє системному архітектору сфокусуватися на процесі проектування саме програмної архітектури. На даний момент існує декілька версій ALD, кожна з яких має спеціальні засоби для окремої предметної області, наприклад: Wright ADL, Rapide ADL, DAOP-ADL, ACME-ADL та ін. Наприклад, нотація Wright ADL застосовується для аналізу комунікаційних протоколів [131] у складі розподілених ПС, натомість Rapide ADL має розвинені можливості для побудови моделей ПС реального часу

[132]. Засоби нотації DAOP-ADL спрямовані на проектування окремих програмних компонентів із урахуванням аспектно-орієнтованого характеру їх взаємодії [133]. Серед представників сімейства мов ADL окремо слід назвати нотацію ACME ADL як таку, що має досить обмежений набір базових конструкцій, який може бути розширений шляхом визначення архітектором додаткових правил функціонування та взаємодії системних елементів ПЗ.

Таким чином, у порівнянні з мовами системного моделювання UML та SysML, сімейство ADL дає можливість відобразити саме архітектурні аспекти моделювання властивостей ПС, що має бути побудована для вирішення певної предметної задачі. Тому в якості адекватної структурно-функціональної нотації для формалізації та подальшого порівняльного аналізу архітектурних моделей ПООТ пропонується використати саме нотацію ACME ADL. Такий підхід не залежить від конкретних засобів програмної реалізації певної ПООТ і дозволяє описати як статичну структуру так і відповідні інтерфейси взаємодії між її елементами.

Базовими поняттями нотації ACME ADL є компоненти (component), порти (port) та конектори (connector), а також такі їх властивості як ролі (role) та взаємодія (interaction). Вони визначаються наступним чином [127, 134, 135]:

– компонент – це елемент опису архітектури, який може логічно об’єднувати декілька функціональних блоків ПС, що виконують певну частину бізнес-логіки системи, кожен компонент повинен мати інтерфейси, які зв’язують його з оточуючим середовищем функціонування ПС;

– порт – це елемент опису архітектури, який слугує інтерфейсом взаємодії компонента із оточуючим середовищем, або з іншими компонентами ПС, при цьому кожен порт може представляти собою як простий інтерфейс, так і складний інтерфейс, який складається із деякої колекції логічно пов’язаних простих інтерфейсів;

– конектор – це елемент опису архітектури, який зв’язує порти різних компонентів, кожен конектор моделює взаємодію між портами і визначає правила, які управляють цією взаємодією за допомогою певних ролей;

– роль – властивість конектора, який визначає учасника взаємодії, представленої цим конектором;

– взаємодія – це властивість конектора, яка визначається за допомогою ролей.

Для коректної побудови архітектурних моделей ПООТ із застосуванням вищенаведених понять нотації ACME ADL слід зазначити, що кожна з окремих ПООТ поєднують у собі на рівні архітектурного моделювання два основних типи компонентів (див. рис. 1.4 – 1.6 у першому розділі дисертації), а саме: базовий об’єктно-орієнтований клас та деяка його функціональна надбудова, логічна сутність якої схожа в усіх ПООТ, але програмна реалізація виконана різними шляхами. Тому для подальшого коректного подання запропонованого підходу, а також з метою розширення базових конструкцій нотації ACME ADL [136], пропонуються наступні визначення [123]:

Визначення 1. *Розширений архітектурний примітив (РАП)* – це мінімально-надлишкова архітектурна компонентна схема, яка потрібна для реалізації взаємодії базових елементів об’єктно-орієнтованого підходу (клас, метод, поле) і додаткових функціональних елементів відповідної ПООТ.

Визначення 2. *Шар (layer)* – є відображенням деякої функціональної вимоги, яка може бути виділена в окремий модуль вихідного коду.

Визначення 3. *Простий порт (single port)* – це інтерфейс взаємодії компонента, для зв’язку його з навколишнім середовищем, який має тільки один вихід або тільки один вхід.

Визначення 4. *Порт з умовою* (*condition port*) – це порт, який має деяку логічну умову і представляє єдину групу логічно пов’язаних інтерфейсів, який має можливість обробляти вхідні повідомлення від декількох різних конекторів.

Визначення 5. *Бінарна взаємодія конектора (binary connector interaction)* – це така схема взаємодії конектора, коли він надає тільки дві ролі наприклад sender → receiver, що відповідають двом портам, які належать двом різним компонентам.

Визначення 6. *Множинна взаємодія конектора (multiple connector interaction)* – це така схема взаємодії конектора, коли він надає більше ніж 2 одночасно можливі ролі.

Визначення 7. *Умовна взаємодія конектора (condition connector interaction)* – це така схема взаємодії конектора, коли він визначає множинну взаємодію, тобто таку, при якій у нього є умова, що може регламентувати порядок обслуговування вхідних та вихідних компонентів.

Використовуючи наведені визначення 1-7 , відповідні РАП можуть бути побудовані як сукупність набору базових об'єктно-орієнтованих компонентів: клас, метод, поле (або *спільних компонентів*) та додаткових (або *варіативних*) компонентів ПООТ, які повинні бути реалізовані в ПС. Компоненти обох цих типів характерні для всіх трьох РАП, які, в свою чергу, утворюють лінійку архітектурних примітивів (ЛАП), відповідно до загального визначення так званої лінійки програмних продуктів (software product line) [137, 138]. Це дозволить, в подальшому, обчислити певні кількісні показники, щоб визначити питомі витрати на реалізацію цих ПООТ.

Для кожної із трьох ПООТ: АОП, ФОП, КОП, розширений архітектурний примітив може розглядатися як аналогія до мовних конструкцій та їх поєднання між собою, де компоненти РАП представляють собою основні конструкції ООП-мов, такі як клас, метод та поле, що містяться у класі; та основні конструкції ПООТ-надбудов, такі як повідомлення та удосконалення (АОП), функціональність та шари (ФОП), шари їх сукупність та контекстний клас (КОП). Порти представляють собою частини сигнатур зазначених елементів, що однозначно їх ідентифікують та конструкції, що утворюються в процесі взаємодії базових

елементів мов ООП із елементами мовних надбудов ПООТ. Конектори у свою чергу забезпечують таку взаємодію, відображаючи приналежність мовних конструкцій до контейнерів таких як клас (ООП), або сукупність шарів (КОП).

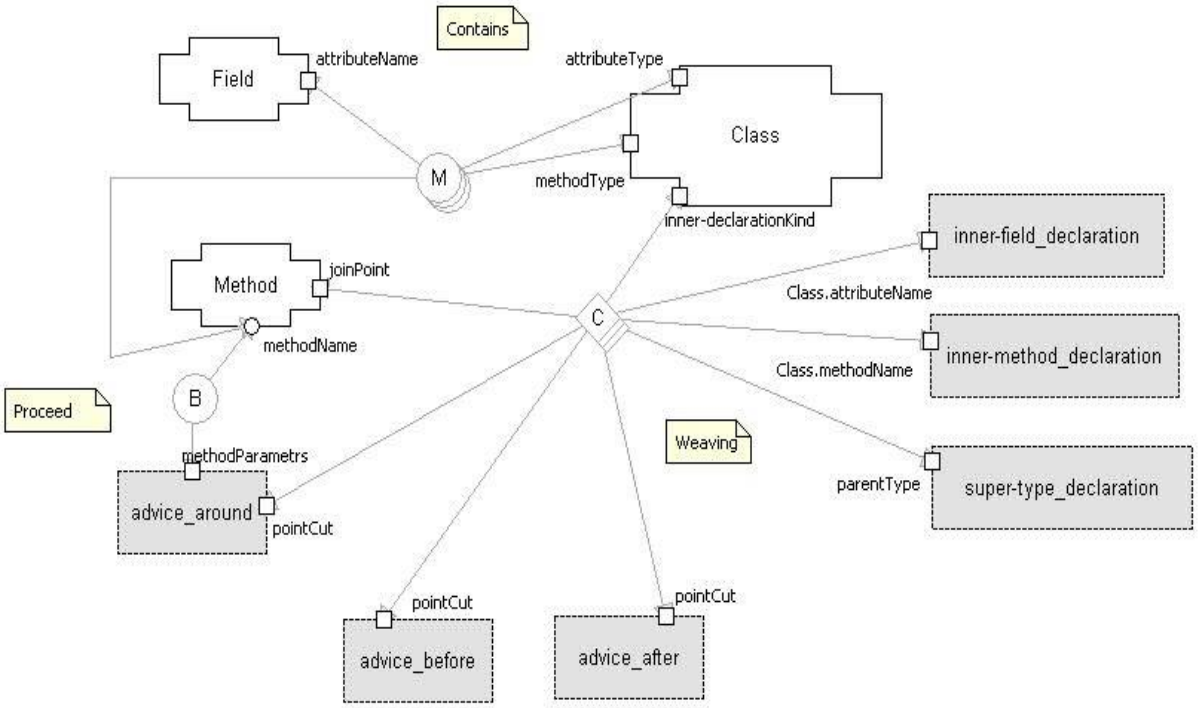


Рисунок 2.6 – Розширений архітектурний примітив для АОП

На рис. 2.6, у відповідності до вербального опису АОП, наведеного у п. 1.3, представлено конструкцію РАП для АОП, до якої входять 3 спільних ООП-компоненти: клас, метод, поле, та 6 варіативних компонентів: повідомлення (advice_before, advice_after, advice_around), внутрішньо-типові декларації (inner-field_declaration, inner-method_declaration, super-type_declaration). Таким чином, РАП для АОП складається із 9 основних компонентів. В свою чергу кожний компонент має колекцію портів, для взаємодії із іншими компонентами:

– клас (Class) має три простих порти. Порт «тип атрибуту» (attributeType), є частиною сигнатури атрибута класу і дозволяє пов’язувати конкретний атрибут, оголошений у класі, із конкретним класом. Порт «тип методу» (mtehodType), що є частиною сигнатури методу і дозволяє пов’язувати

конкретний метод, оголошений у класі, із конкретним класом. Та порт «тип інтродукції» (inner-declarationKind), є абстракцією точки з'єднання, конкретної інтродукції із конкретним класом;

– поле (Field) має один простий порт «attributeName», що також є частиною сигнатури атрибута;

– метод (Method) має два порти, один з яких порт з умовою – «methodName», є частиною сигнатури метода і дозволяє асоціювати конкретний метод із конкретним класом, а також взаємодіяти із компонентом «повідомлення» (advice_around). Та один простий порт «точка з'єднання» (joinPoint), що дозволяє здійснювати процес зв'язування повідомлень та методів;

– варіативні компоненти логічної групи «повідомлення» (advice_around, advice_before, advice_after), мають по одному простому порту «зріз» (pointCut), що дає можливість утворюватися «точкам з'єднання» для процесу зв'язування;

– варіативні компоненти логічної групи «інтродукції» (inner-field_declaration, inner-method_declaration, super-type_declaration) мають по одному простому порту «ім'я поля», «ім'я метода», «супер-тип» (Class.attributeName, Class.methodName, superType). Ці порти дають змогу установити з'вязок із конкретним класом, для створення нових полів і методів, або зміни в дереві наслідування класу.

Взаємодія між компонентами здійснюється завдяки конекторам:

– множинна взаємодія конектора, який вказує на приналежність (Contains) компонентів «Поле» та «Метод» до конкретного класу «Клас», тобто на рівні ОО-мови він вказує що поле і метод містяться у класі, або належать йому, з відповідними унікальними сигнатурами;

– бінарна взаємодія конектора «продовження» (Proceed), реалізує специфічну взаємодію між відповідним компонентами;

– умовна взаємодія конектора «компоновника» (Weaving), реалізує зв'язування відповідних повідомлень та інтродукцій із усіма базовими

компонентами завдяки зрізам, точкам з’єднання. Специфіка конектора у фільтрації взаємодії компонентів advice_before, advice_after із advice_around, яка одночасно не можлива.

РАП для ФОП (див. рис. 2.7) також включає 3 базових компоненти та 4 варіативні компоненти: внутрішньо-типові декларації (inner-field_declaration, inner-method_declaration), функціональність або властивість (feature), equation (компонент композиції).

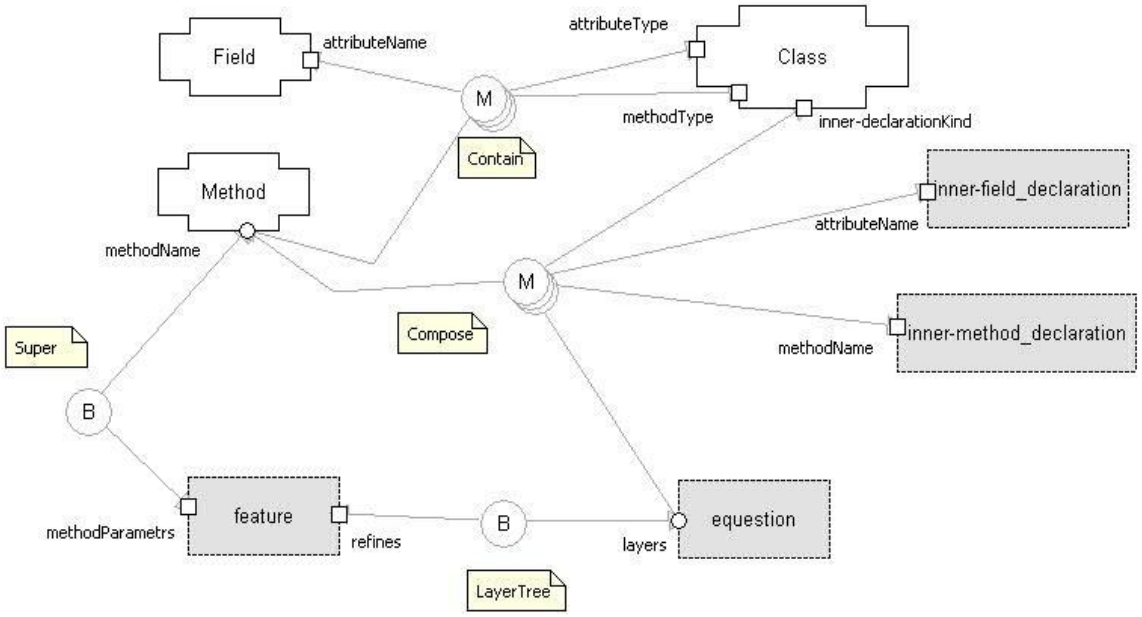


Рисунок 2.7 – Розширений архітектурний примітив для ФОП

Таким чином, РАП для ФОП складається із 7 основних компонентів. В свою чергу кожний компонент має колекцію портів, для взаємодії із іншими компонентами:

– клас (Class) має три простих порти, що аналогічні портам у такому ж компоненті у РАП АОП;

– поле (Field) має один простий порт також аналогічний такому ж компоненту в РАП АОП;

– метод (Method) має тільки один порт з умовою, що аналогічний такому ж компоненту у РАП АОП;

– варіативні компоненти логічної групи «інтродукції» також мають по одному порту аналогічно із відповідними портами у РАП АОП;

– функціональність (Feature) має два простих порти. Порт «параметри методу» є частиною сигнатури метода, та аналогічний такому ж порту у РАП АОП. Завдяки простому порту «вдосконалення» (refine), даний компонент має змогу установлювати досить складну взаємодію через проміжні порти іншого компоненту із цільовими методами ОО-класу;

– реєстр (Equation) – компонент із складним портом з умовою, що представляє собою колекцію інтерфейсів шарів (layers), що відповідають за зв’язування

конкретної функціональності із конкретним методом конкретного класу.

Взаємодія між компонентами здійснюється завдяки конекторам:

– множинна взаємодія конектора, який вказує на приналежність (Contains) така ж як у РАП АОП;

– бінарна взаємодія конектора «супер» (Super), діє по аналогії із взаємодією конектора «продовження» (Proceed) РАП АОП;

– множинна взаємодія конектора «композиція» (Compose), схожа на взаємодію конектора «компоновник» (Weaver), крім відсутності фільтра, тому що в ФОП усі компоненти можуть взаємодіяти одночасно без виключень;

– бінарна взаємодія конектора «дерево слоїв» (LayerTree), зв’язує функціональність, що має удосконалити метод, із функціональним шаром в якому вона належить.

Технологія ФОП дозволяє деталізувати базові ООП-класи, розширюючи їх первинні методи та доповнюючи їх новими, за допомогою механізму композиції шарів, кожний з яких відповідає новій властивості ПС.

РАП для КОП (див. рис. 2.8) також включає 3 базових компоненти та 3 варіативні компоненти: контекстний клас (contextClass), шар (layer), сукупність шарів (layers) – компонент, який включає сигнатури всіх оголошених у ПС шарів. Сигнатури цього компоненту мають бути імпортовані в базові класи для їх подальшого визначення та використання.

Порти базових компонентів у цьому РАП аналогічні портам, що належать компонентам РАП АОП та ФОП. Варіативні компоненти: «шар» (Layer), що містить додаткову функціональність, та «сукупність шарів» (Layers) взаємодіють через відповідні порти «ім'я шару» (layerName) та «визначення шару» (layerDefinition) та взаємодію конектора приналежності (Contains), що аналогічний з РАП АОП та ФОП. Компонент «контекстний клас» (ContextClass), відповідає за взаємодію шарів із класом шляхом активації та деактивації шарів в залежності від контексту виконання. По суті він є конектором, який за правилами ALD може бути представлений як компонент на більш деталізованому рівні абстракції.

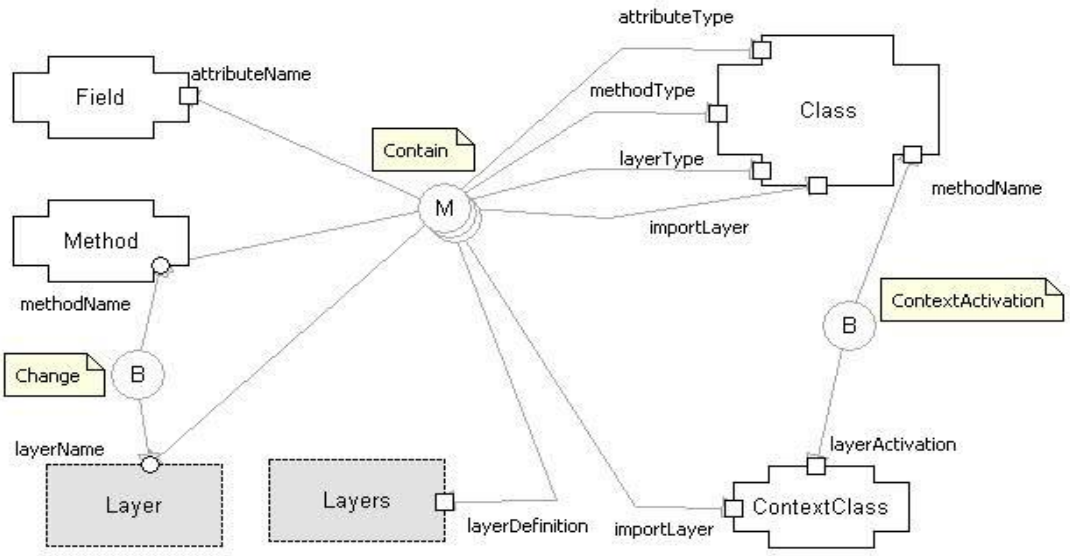


Рисунок 2.8 – Розширений архітектурний примітив для КОП

Технологія КОП надає можливість підключати та відключати шари динамічно під час виконання програми, в залежності від контексту і підтримує можливість визначення шарів в тілі базового ООП-класу.

Таким чином, для остаточного вирішення *Задачі 2*, що поставлена в рамках запропонованого у п. 2.1 знання-орієнтованого підходу, на базі побудованих РАП можливо реалізувати відповідний метод обчислення витрат на їх реалізацію, опис якого наведено у наступному розділі дисертації.

2.5. Розробка метрик для комплексної оцінки рівня присутності наскрізної функціональності (НФ) у програмному коді УПС

Відповідно до запропонованого знання-орієнтованого підходу для вирішення загальної проблеми визначення ефективності використання ПООТ в процесі супроводу УПС, для вирішення *Задачі 3*, з переліку *Задач 1 – 4*, який приведено у п. 2.1 цього розділу, а саме: для оцінки рівня присутності НФ у програмному коді УПС, необхідно більш детально розглянути природу та поведінкові аспекти НФ і запропонувати метрики, що дозволять кількісно визначати рівень присутності НФ в УПС.

На сьогоднішній день існує багато досліджень, присвячених проблемі НФ взагалі, та варіативності її проявів у вихідному коді ПС зокрема. В літературі досліджуються властивості НФ, наводяться різні варіанти її типізації відповідно до цих властивостей та пропонуються відповідні засоби реакції на ті чи інші типи НФ.

Найпростіший підхід до визначення типу НФ – це підхід, що орієнтується на природу НФ: гомогенна або гетерогенна НФ [45] (див. п. 1.2 попереднього розділу).

В дослідженні [139] пропонується розділення НФ на відповідні шаблони (їх 13) які розбиті на 4 категорії:

- «плоскі шаблони НФ» (flat crosscutting patterns) - до цієї групи входять шаблони, що відображають «внесок» (dedication) відповідного класу в реалізації НФ;
- «шаблони наслідування» (inheritance-wise concerns) - до цієї групи входять шаблони, що відображають ураження НФ дерева наслідування (inheritance tree);
- «шаблони зв'язку» (communicative concerns) - до них входять шаблони, що відображають ураження НФ зв'язків між класами;
- «інші шаблони» (miscellaneous), що не увійшли до перших трьох груп.

В рамках цього дослідження запропоновані стратегії щодо виявлення таких шаблонів НФ та даються рекомендації, яким чином можливо усунути реалізації НФ, що віднесені до конкретного шаблону за допомогою АОП.

У роботах [140, 141, 142] запропоновано розділяти НФ на сорти, всього таких сортів представлено 13. Кожен сорт НФ характеризується п'ятьма атрибутами, що відображають його «назву» (name), «намір» (intent), «успадковану реалізацію» (legacy implementation), «бажану реалізацію» (desired implementation), «варіанти» (instances), наприклад:

- *назва* (сорту НФ) – «Прошарок переспрямування»(Redirection layer);
- *намір* (опис) – НФ визначає інтерфейсний прошарок для об'єкта та перенаправляє виклики до цього об'єкта;
- *успадкована реалізація* (ООП) – визначення проміжного прошарку (декоратор або адаптер), та додання методів до цього прошарку, що перенаправляють їх виклики;
- *бажана реалізація* (АОП) – зріз (pointcut) та порада заміщення (around advice);
- *варіанти* – шаблон «Декоратор», шаблон «Адаптер» [19].

Для кожного сорту НФ надається порада щодо усунення її за допомогою АОП, тобто в рамках цих робіт, бажана реалізація є аспектно-орієнтована реалізація.

Зазначені вище способи класифікації НФ: шаблони, сорти з точки зору команди супроводу УПС можуть давати загальне розуміння присутності НФ, відноситься до деякого класу, та деякі можливі стратегії її видалення. Але ці класи не дають кількісного уявлення про складність ураження системи НФ. З огляду на це в рамках даного дослідження пропонується додатковий клас, що має розрізняти різні реалізації НФ за ступенем поширення її в УПС: «слабке ураження», «ураження середньої тяжкості», «сильне ураження». Ступінь ураження УПС можна виразити через коефіцієнт питомої ваги НФ (Crosscutting Functionality Ratio – CFR), що запропоновано у [46]:

$$CFR = \frac{C_{cf}}{N}, \quad \text{CFR} \in [0;1], \tag{2.20}$$

де CFR – коефіцієнт питомої ваги НФ в УПС,

C_{cf} – кількість класів базової бізнес-логіки УПС, де присутній вихідний код НФ,

N – загальна кількість класів в УПС.

Якщо $CFR = 1$, це свідчить про те, що НФ уражені всі класи УПС. Якщо $CFR = 0$, присутності НФ в системі не виявлено. При цьому граничні показники коефіцієнта питомої ваги НФ, пропонується обирати за принципом звичних знань для нормалізованих показників, згідно із [143], що наведені у таблиці 2.1:

Таблиця 2.1.

Граничні значення для нормалізованих показників

Граничне значення	Семантичне значення
0.25	Одна чверть
0.33	Одна третина
0.5	Половина
0.66	Дві третини
0.75	Три чверті

Таким чином, граничні значення діапазонів ураження УПС наскрізною функціональністю будуть наступні:

- якщо значення метрики $0 \leq CFR < 0.33$ – то дана УПС має слабке ураження, тобто НФ присутня менше ніж у третині її класів;
- якщо значення метрики потрапляє в діапазон $0.33 \leq CFR < 0.75$ – то дана УПС має ураження середньої тяжкості, тобто НФ присутня у трьох чвертях її класів;
- якщо значення метрики $0.75 \leq CFR \leq 1$ – то дана УПС має сильне ураження, тобто НФ присутня майже в усіх її класах;

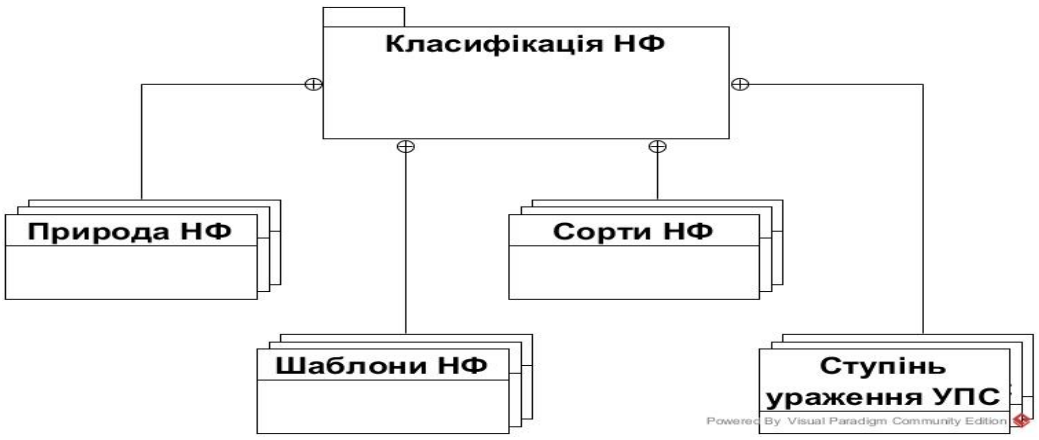


Рисунок 2.9 – Класифікація наскрізної функціональності

Рисунок 2.9 в термінах діаграми класів, нотації UML 2.0 [124] відображає існуючі (запропоновані в інших роботах) класи НФ, та з огляду на їх недоліки до існуючої класифікації додано новий клас за ступенем ураження УПС наскрізною функціональністю [46].

Запропонований для цього вираз (2.20), дає змогу досить швидко оцінити яка питома вага НФ в заданій УПС, що дає змогу визначати підсистеми, складність супроводу яких зростає. Але на ряду із цим вираз (2.20) не дає розуміння повної картини ситуації, з огляду основні властивості НФ (див. п.п. 1.2 попереднього розділу). Для цього потрібне залучення додаткових метрик НФ.

У дослідженні [139] наводиться огляд та робиться спроба систематизації деяких метрик (запропонованих у інших роботах, наприклад [144, 143, 145]), що вимірюють різні показники НФ та її взаємодію із класами основної бізнес-логіки. Зокрема такі як: *CDO* – поширення НФ серед методів (Concern diffusion over operations), або *CDLOC* поширення НФ серед строк програмного коду (Concern diffusion over lines of code) *CDLOC*, які були запропоновані у [145]. Зазначені метрики із різним ступенем гранулярності вимірюють заплутування (tangling) НФ (див. п.п. 1.2. розділу 1).

Метрика *CDO* інкрементує кількість методів, пов’язаних із НФ, та вимірюється у натуральних числах, і лежить в діапазоні $0 \le CDO < +\infty$.

Метрика *CDLOC* інкрементує кількість функціональних перемикачів (switch) для кожного компоненту, що реалізує дану функціональність. Перемикач, у даному контексті, – це погранична пара рядків вихідного коду, де перший рядок реалізує НФ, а другий – іншу функціональність (не наскрізну, або наскрізну), та вимірюється у натуральних числах, і лежить в діапазоні $0 \le CDLOC < +\infty$.

У роботі [144] запропоновані метрики «розміру» (Size), «стику» (Touch), «розповсюдження» (Spread) та «фокусу» (Focus), що пов’язані із розрахунками розподілення НФ (див. рис. 2.9).

В [146] запропоновано декілька метрик, та також присутнє порівняння їх із уже відомими. Одна із запропонованих у зазначеній роботі метрик є ступеня розсіювання (Degree of Scattering) *DS*, що вимірює ступінь розсіювання НФ серед класів УПС. Вона показує на скільки таке розсіювання рівномірне:

$$DS = 1 - \frac{N}{N-1}\sum_{n=2}^{N}\left(\frac{LOC(cf,n)}{LOC(cf)} - \frac{1}{N}\right)^2,\ DS \in [0;1], \tag{2.21}$$

де N – кількість класів бізнес-логіки, $N > 1$,

LOC(cf,n) – кількість строк коду у класі *n*, що відносяться до НФ,

LOC(cf) – загальна кількість строк коду, що відноситься до НФ.

Якщо $DS = 1$, це свідчить про те, що НФ рівномірно розсіяна серед уражених їй класів УПС. Якщо ж $DS = 0$, то така НФ повністю ізольована. Ця метрика є досить придатною для подальших розрахунків, за виключенням одного недоліку – вона не відображає кількість класів, що уражені в УПС. Тобто не є зрозумілим, у якій саме частині УПС присутня НФ, яка наприклад рівномірно розсіяна.

Беручи до уваги різні аспекти НФ, що можуть бути виміряні за допомогою зазначених вище метрик, в рамках даного дослідження пропонується поєднати деякі із них (2.20) та (2.21). Таким чином, комбінована метрика буде показувати на скільки рівномірно розсіяна (scattered) НФ серед уражених їй класів УПС [147]:

$$CFL = DS \cdot CFR,\ CFL \in [0;1], \tag{2.22}$$

де *CFL* – ступінь присутності НФ (Crosscutting Functionality Level),

DS – ступінь розсіювання НФ (Degree of Scattering), вираз (2.21),

CFR – коефіцієнт питомої ваги НФ (Crosscutting Functionality Ratio), вираз (2.20).

Якщо *CFL* = 1, це свідчить про те, що НФ рівномірно розсіяна серед усіх класів УПС, тобто уражені усі класи. Якщо *CFL* = 0, то НФ повністю ізольована в одному класі.

Таким чином, в даному підрозділі були розглянуті способи класифікації НФ та метрики, на базі яких було запропоновано комбіновану метрику визначення ступеня присутності та характеру розповсюдженості НФ в УПС, що дає досить повну інформацію відносно стану ураженої УПС.

Висновки по розділу 2

У даному розділі дисертаційної роботи:

1. Проаналізовані та формалізовані чинники, що впливають на процес оцінки ефективності використання пост об’єктно-орієнтованих технологій (ПООТ), визначені основні характеристики необхідних для цього інформаційних ресурсів, і запропоновано знання-орієнтований підхід для вирішення задачі визначення ефективності використання ПООТ при супроводі відповідної успадкованої програмної системи (УПС).
2. Розглянуто моделі та метрики оцінки структурної складності різних типів УПС, запропоновано їх класифікацію та визначено групу метрик, які адекватно обчислюють структурну складність об’єктно-орієнтованих УПС.
3. На основі поєднання позитивних властивостей кількісних та якісних методів аналізу складності вимог до ПЗ запропоновано логіко-лінгвістичний підхід до оцінки стану вимог до УПС в процесі її супроводу.
4. Побудовано архітектурні моделі окремих ПООТ, які уможливлюють отримання в подальшому кількісних оцінок витрат, які пов’язані із застосуванням ПООТ для модифікації УПС.
5. Проаналізовано підходи до оцінки характеристик НФ та запропонована комплексна метрика для оцінки ступеня та характеру ураження УПС

наскрізною функціональністю.

Нові наукові результати, викладені в цьому розділі, опубліковані в роботах [123, 46, 147, 86].

РОЗДІЛ 3.
РОЗРОБКА МОДЕЛЕЙ, МЕТОДІВ ТА ТЕХНОЛОГІЧНИХ ПРОЦЕДУР ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ПОСТ ОБЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ

У цьому розділі, на основі запропонованих у попередньому розділі методологічних принципів побудови модельно-технологічного інструментарію для оцінки ефективності застосування ПООТ в процесах супроводу УПС, наведені результати розробки відповідних моделей, методів та технологічних процедур, які дозволяють реалізувати цю задачу.

У першому параграфі, на основі запропонованого знання-орієнтованого підходу, розроблена алгоритмічна модель процесу експертної оцінки ефективності ПООТ при супроводі УПС.

Другий параграф містить опис методу визначення типу УПС із урахуванням таких чинників як структурна складність їх програмного коду, а також рівень складності та динаміки змін у функціональних вимогах користувачів до таких систем.

У третьому параграфі наведено опис метрик та алгоритмів визначення питомих витрат на проведення структурної модифікації УПС із застосуванням певної ПООТ на основі їх архітектурних моделей.

Зміст четвертого параграфу присвячено висвітленню питань розробки процедури комплексної оцінки ефективності застосування ПООТ на основі застосування методів нечіткої логіки.

У п’ятому параграфі представлена загальна схема прикладної інформаційної технології для автоматизації виконання всіх основних етапів запропонованого підходу.

3.1. Розробка алгоритмічної моделі процесу експертної оцінки ефективності застосування ПООТ

Запропонований у другому розділі дисертації (див. п. 2.1) знання-орієнтований підхід до побудови модельно-технологічного інструментарію для дослідження властивостей ПООТ передбачає, що при розгляді особливостей процесів їх розробки та подальшого використання при супроводі УПС необхідно враховувати взаємодію великої кількості чинників, які в свою чергу можуть бути визначені шляхом аналізу досить складних та слабоформалізованих інформаційних об'єктів. У формалізованому вигляді взаємодія цих чинників подана шляхом формулювання *Тверджень 1-5* (див. п.п. 2.1 попереднього розділу), а їх геометрична інтерпретація на концептуальному рівні подана у вигляді метафори багатовимірного інформаційного простору, що представлена на рис. 2.1. Таким чином, зважаючи на ці особливості процесу визначення ефективності застосування ПООТ при супроводі складних УПС, для вирішення цієї задачі може бути запропонована побудова відповідної алгоритмічної моделі, поняття про яку ефективно застосовується в таких роботах, як, наприклад, [148, 149]). Ця модель з методологічної точки зору поєднує в собі окремі методи, алгоритми та метрики, що були визначені у п.п. 2.2 – 2.5 другого розділу дисертації для вирішення переліку комплексних *Задач 1-4* , а також слугує основою для розробки відповідних інструментальних засобів, які потім мають бути інтегровані у прикладну інформаційну технологію.

Запропонована алгоритмічна модель (*algorithmic model - AM)* ***AM*** може бути подана у наступному вигляді [150]

$$AM = \langle InfoSpace, WorkFlow(Methods), CFMetrics \rangle, \qquad (3.1)$$

де *InfoSpace* – багатовимірний інформаційний простір (Information Space) алгоритмічної моделі;

WorkFlow(Methods) – сукупність алгоритмів (workflow), які імплементують методи експертної оцінки (methods) відповідних характеристик ПООТ;

CFMetrics – колекція метрик (metrics) визначення ступеню присутності наскрізної функціональності (crosscutting functionality – CF) у програмному коді модулів УПС.

Для подальшого структурування інформаційного базису моделі з виразу (3.1) слід враховувати вирази (2.1) – (2.5), які відповідно формалізують визначення таких понять як

- множина існуючих типів УПС (множина ***S***),
- множина різних ПООТ (множина ***T***),
- множина можливих значень коефіцієнту ефективності застосування ПООТ при супроводі певної УПС (множина $\boldsymbol{K_{Eff}}$).

Тоді багатовимірний інформаційний простір *InfoSpace* алгоритмічної моделі ***AM*** може бути подано у вигляді наступного кортежу:

$$InfoSpace = \langle \Pi_1, \Pi_2, \Pi_3 \rangle, \tag{3.2}$$

Окремі складові кортежу (3.2), в свою чергу, структуруються у наступний спосіб:

Π_1 – це інформаційний простір, у якому мають бути визначені оцінки ефективності застосування окремих ПООТ, при цьому

$$\Pi_1 \subseteq S \times T \times K_{Eff}\ , \tag{3.3}$$

де множини ***S, T*** і $\boldsymbol{K_{Eff}}$ вже були визначені вище;

$П_2$ – це інформаційний простір, у якому мають бути визначені можливі типи УПС, він подається як

$$П_2 \subseteq StructuralComplexity \times RequirementRank , \quad (3.4)$$

де *StructuralComplexity* – це множина можливих значень такої характеристики УПС, як її структурна складність (structural complexity), а *RequirementRank* – це множина значень такої характеристики УПС, як ранг функціональних вимог, що надходять від користувачів УПС і мають бути враховані в процесі її супроводу;

$П_3$ – це інформаційний простір, у якому визначені можливі значення рангу функціональних вимог *RequirementRank*, а саме

$$П_3 \subseteq RequirementWeight \times RequirementPriority , \quad (3.5)$$

де *RequirementWeight* – це множина можливих значень ваги (або складності) відповідної функціональної вимого до УПС в процесі її супроводу, а *RequirementPriority* – множина можливих значень такої характеристики вимоги, як пріоритет її реалізації. Ці питання більш детально викладені у наступних підрозділах тексту дисертаційної роботи.

3.2. Розробка метода визначення стану УПС у процесі супроводу

На основі вже методологічно обґрунтованих обраних експертних методів і кількісних метрик для оцінки структурної складності УПС (див. п. 2.2 попереднього розділу) і запропонованого логіко-лінгвістичного підходу до моделювання та аналізу стану вимог до УПС (див. п. 2.3) можливо розробити обчислювальний метод та відповідні алгоритми для визначення стану УПС у процесі супроводу.

Для візуального представлення цього методу пропонується застосувати геометричний підхід [151, 152], який використовує характеристичний простір ознак і дає більш наглядну трактовку критеріям, які утворюють простір $П_2$ – це «Тип Системи» (див. вираз (3.4)), що відповідно включає узагальнену характеристику вимог – *ранг ФВ* (Requirement Rank), та початкову об'єктно-орієнтовану *структурну складність* УПС. В свою чергу, *ранг ФВ* – це складний критерій, який геометрично представляє собою простір $П_3$.

Простір $П_3$ - «Ранг вимог», - представляє собою декартовий добуток двох характеристик (див вираз (3.5)): *складність* вимоги та *пріорітет* вимоги, які необхідно визначити для оцінки загального рангу ФВ. Його графічна інтерпретація приведена на рис. 3.1, де чорними точками схематично зображено окремі вимоги до УПС.

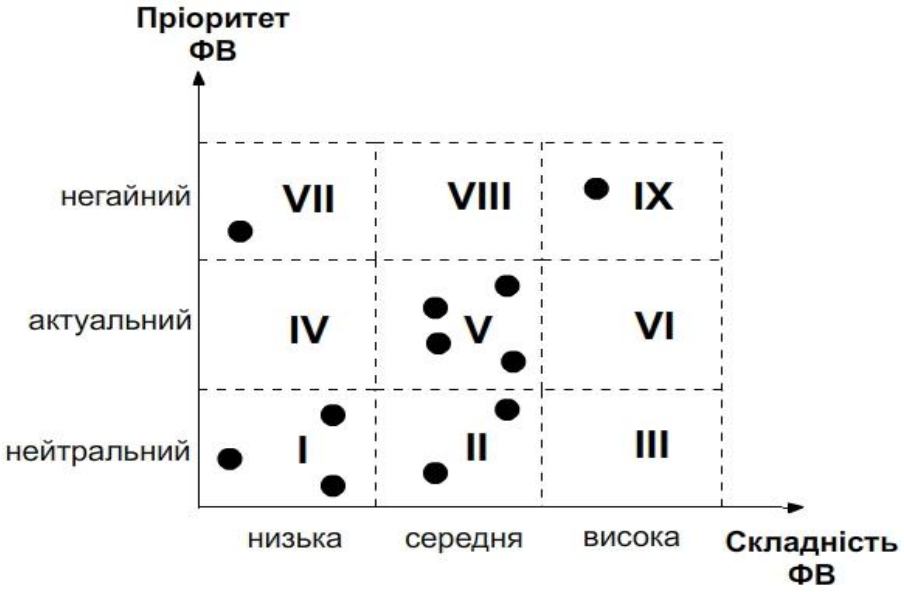


Рисунок 3.1 – Простір $П_3$ «Ранг вимог»

У [153] запропоновано метод для визначення загального рангу ФВ та переходу із простору $\boldsymbol{П_3}$ «Ранг вимог» до простору $\boldsymbol{П_2}$ «Тип УПС». Він передбачає послідовне виконання 2-х наступних алгоритмів.

Алгоритм 1. *Відображення простору «Ранг вимог» у простір «Тип ПС»*

Крок 1. Визначити початкові зони складності (ЗС) у просторі «Ранг вимог». Маємо n рівнів складності ФВ та m рівнів пріоритету ФВ. Кількість ЗС – #ComplexityZone, обчислюється наступним виразом:

$$\#ComplexityZone = n \cdot m \tag{3.6}$$

Крок 2. Згрупувати усі початкові зони складності у необхідну кількість груп, за допомогою алгоритму кластеризації k–середніх (див. наприклад у [154]). Для зручності подальших розрахунків у просторі «Тип УПС» таких кластерів слід обрати 3.

Крок 3. Визначити вагові коефіцієнти w_i для кожного кластеру складності, за допомогою методу аналізу ієрархій [155]: $w_1 = 0{,}16$; $w_2 = 0{,}34$; $w_3 = 0{,}5$.

Крок 4. Розрахувати загальний ранг вимог УПС за допомогою виразу:

$$RequirementRank = \sum_{i=1}^{k} w_i (\#R)_i, k = 3, \tag{3.7}$$

де $\#R$ – кількість вимог у відповідному кластері складності;

w_i – вага кластеру складності, отримана на кроці 3,

k – кількість кластерів складності.

Крок 5. Визначити межі шкали вимірювання для ранга ФВ:

$$RequirementRank \begin{cases} [Rank_d; Rank_{\mu 1}) \\ [Rank_{\mu 1}; Rank_{\mu 2}), \\ [Rank_{\mu 2}; Rank_u] \end{cases} O_\mu(Rank_m) = \{x : |x - Rank_m| < \mu\}, \tag{3.8}$$

де $Rank_o$ – відповідні межі для інтервальної шкали вимірювання: *нижня, середня, висока* межі;

$O_\mu(Rank_m)$ – μ-окіл для середньої межі інтервальної шкали.

Крок 6. Перехід від простору «Ранг вимог» до простору «Тип ПС», побудова лінгвістичної змінної «Ранг ФВ» (це поняття більш детально вже було розглянуто в п. 2.3 попереднього розділу).

На *Кроці 2* запропонованого алгоритму проводиться кластеризація початкових зон складності. Це потрібно для зручності подальших розрахунків, див. рис. 3.2.

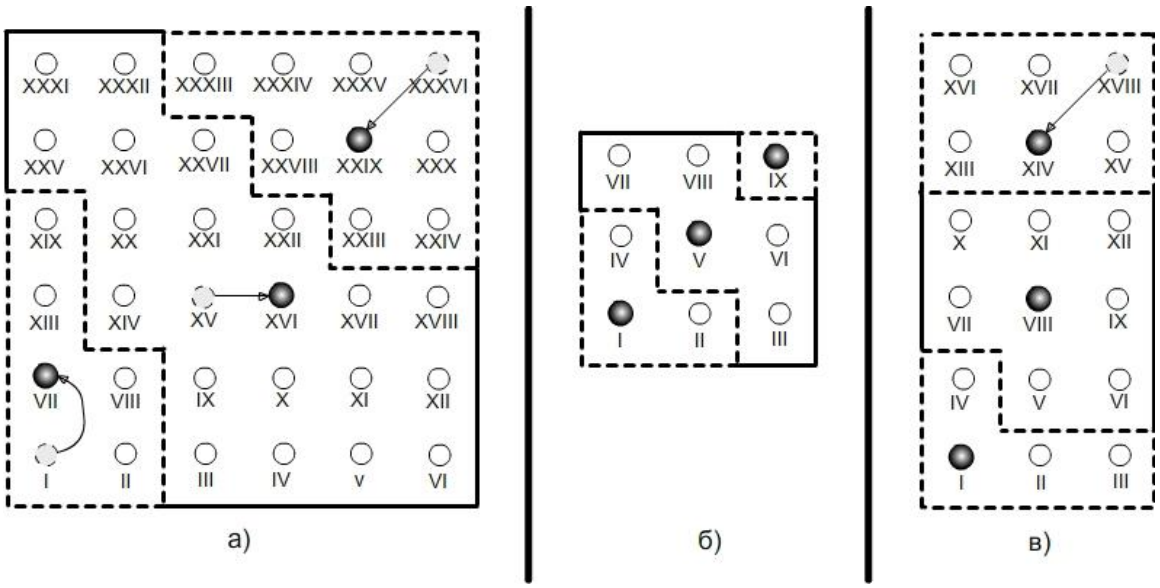


Рисунок 3.2. – Варіанти групування початкових зон складності у просторі «Ранг вимог»

Кількість початкових зон визначається кількістю термів у лінгвістичних змінних «Пріоритет ФВ», «Складність ФВ». В рамках цього дослідження таких термів для кожної лінгвістичної змінної було обрано по 3, виходячи з кількостей відповідних параметрів у функціональності більшості сучасних систем управління вимогами (СУВ). Тобто прості «Ранг вимог» складається із 9 початкових зон складності.

Інший принцип формування кількості початкових зон складності може давати більше їх число, як показано на рис. 3.2, де пунктирними лініями показані границі трьох кластерів складності для, трьох прикладів: а) простір «Ранг вимог» складається із (6х6) початкових зон складності; б) простір «Ранг вимог» складається із (3х3) початкових зон складності; в) простір «Ранг вимог» складається із (3х6) початкових зон складності. Колами на цьому рисунку показані відповідні зони складності (з їхніми координатами у просторі «Ранг вимог»). Сірими колами з пунктирними границями відображені первинні центри кластерів, чорні кола відображають кінцеві центри кластерів (відповідно до алгоритму кластеризації *k*-середніх, див наприклад [154]).

Таким чином, представлена процедура кластеризації дозволяє згрупувати довільну кількість зон складності у необхідну кількість кластерів.

Лінгвістична змінна, отримана на *Кроці 6* запропонованого алгоритму для рангу вимог, відповідно до загального її визначення (див. вираз (2.18)), приймає наступний вигляд:

X – «Ранг ФВ»;

T(X) – {*низький, середній, високий*},

U – множина числових значень *рангу вимог*, отримана в результаті використання запропонованого алгоритму переходу,

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функція відповідності Харінгтона.

Для оцінки початкової об’єктно-орієнтованої структурної складності УПС пропонується використати наступний алгоритм, що базується на визначенні кількісних показників ООП-метрик (див. вираз (2.15)).

Алгоритм 2. *Оцінка початкової структурної складності УПС*

Крок 1. Обчислити кількісні показники за метриками ОО-коду (див. вирази (2.6) – (2.13), (2.15)).

Крок 2. Відповідно до виразу (2.16) визначити вагу w_i для кожної із метрик.

За методом MAI [155]: $w_1 = 0{,}04$; $w_2 = 0{,}01$; $w_3 = 0{,}06$; $w_4 = 0{,}11$; $w_5 = 0{,}13$; $w_6 = 0{,}22$; $w_7 = 0{,}22$; $w_8 = 0{,}21$.

Крок 3. Визначити процентне співвідношення класів, що потрапляють до А:«простої», В: «середньої», С: «складної» зони, відповідно до інтервалів меж, для зазначених метрик коду [143, 156].

Крок 4. Назначити вагові коефіцієнти p_j для категорій складності, описаних на кроці 3 цього алгоритму: $p_1 = 0{,}1$; $p_2 = 0{,}3$; $p_3 = 0{,}6$.

Крок 5. Розрахувати показник складності за конкретною метрикою *M*:

$$CM = \sum_{k=1}^{3} p_k \cdot Z_k, \tag{3.9}$$

де *CM* – показник складності метрики (ComplexityMetric),

p_k – ваговий коефіцієнт для зони складності,

Z_k – процентна частка одиниці розрахунку (методи, класи), що належать до кожної із трьох зон складності.

Крок 6. Розрахувати максимальне кількісний показник для кожної із категорій складності, за принципом: «100% класів(методів) потрапляє до зони складності»:

$$Z_k = p_k \cdot 100\%, k \in [1;3], \tag{3.10}$$

де Z_k – зона складності,

p_k – ваговий коефіцієнт для зони складності.

Тобто: $Z_A = 10$; $Z_B = 30$; $Z_C = 60$.

Крок 7. Визначити граничні значення для зон структурної складності УПС, підставивши у вираз (2.16) відповідні максимальні значення зон складності,

отримані на *Кроці 6*. Таким чином, граничні значення структурної складності визначаються виразом (3.11):

$$StructuralComplexity \begin{cases} [0;10) \\ [10;30) \\ [30;60] \end{cases}, \tag{3.11}$$

Крок 8. Розрахувати фактичну структурну складність УПС, на базі виразів (3.11) та (2.16).

Крок 9. Виразити кількісний показник структурної складності через лінгвістичну змінну: «Структурна Складність», яка має наступний вигляд:

X – «Структурна Складність»;

T(X) – {*проста*, *середня*, *складна*},

U – множина числових значень *структурної складності*, отримана в результаті використання алгоритму оцінки складності,

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функція відповідності Харінгтона [122].

В результаті послідовного виконання запропонованих алгоритмів можливо однозначно визначити позицію УПС у просторі $П_2$ «Тип УПС». На рис. 3.3 представлена графічна інтерпретація лінгвістичної змінної «Ранг ФВ» у просторі «Тип ПС», яка позначена на вісь ординат.

Вісь абсцис у просторі $П_2$ відображає лінгвістичну змінну «Структурна Складність» УПС, таким чином, на рис. 3.3 білою точкою позначена УПС, що однозначно належить до IV типу, тобто має «середній» ранг ФВ, та «просту» структурну складність.

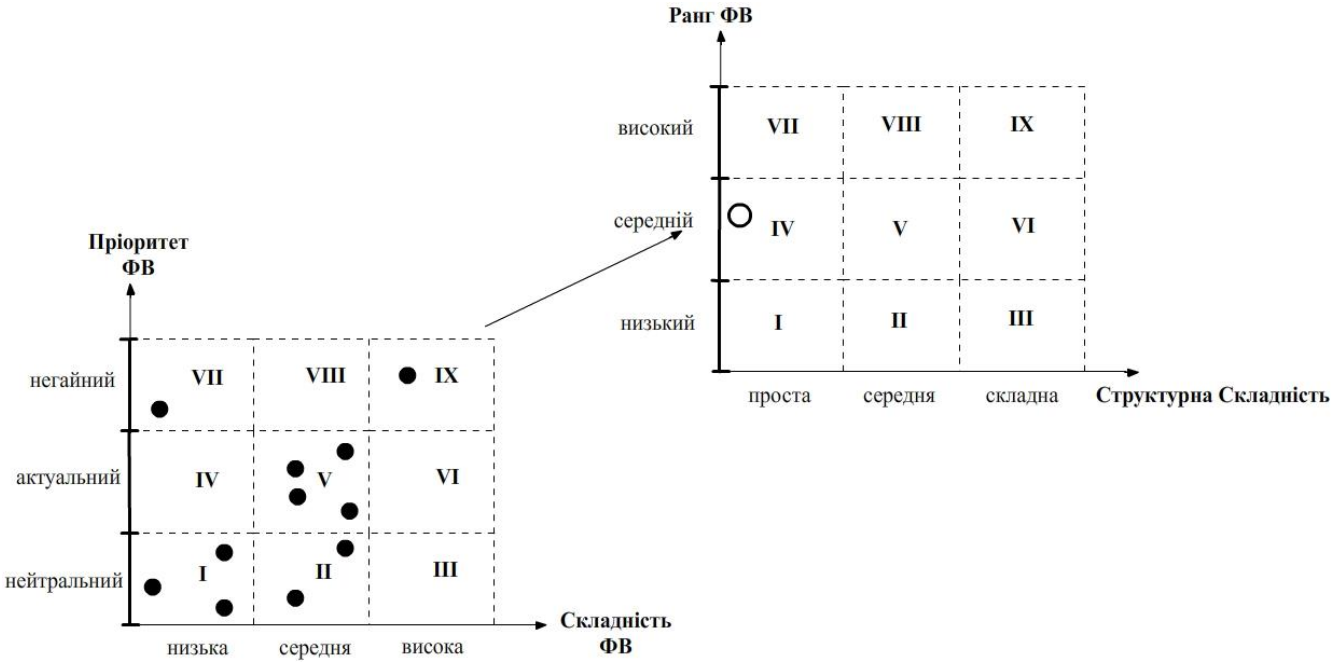


Рисунок 3.3. – Перехід із простору «Ранг вимог» до простору «Тип ПС»

Таким чином, запропонований метод дозволяє однозначно визначити тип даної УПС та позиціонувати її у просторі $П_2$ «Тип ПС», який належить до інформаційного базису запропонованої алгоритмічної моделі (3.1) загального процесу визначення ефективності застосування ПООТ при супроводі УПС.

3.3 Розробка метрик та алгоритмів визначення питомих витрат на застосування ПООТ на основі їх архітектурних моделей

Визначення оцінки витрат на програмну реалізацію модифікованої структури УПС із використанням ПООТ передбачає можливість обчислення питомої складності розширених архітектурних примітивів (РАП) [123], докладний опис яких наведено у п. 2.4. Для обчислення складності цих архітектурних моделей необхідно визначити складність кожної групи базових архітектурних елементів, що є складовими цих моделей: *компонент (component), порт (port), конектори (connector)*, і для цього може бути використаний адитивний критерій згортки [157].

Загальна складність групи компонентів (*ComponentComplexity*) у запропонованих моделях РАП (див рис. 2.6 – 2.8), відповідно до [123], визначається наступним виразом:

$$ComponentComplexity = \sum_{c=1}^{2} w_c \cdot \#(C)_c; \ \sum_{c=1}^{2} w_c = 1, \tag{3.12}$$

де $\#(C)_c$ – це кількість компонентів. Індекс c – позначує приналежність складових до одного із двох типів: *компоненти ООП–бази* (тобто це основні класи ПЗ) та *компоненти ПООТ–надбудови* (це додаткові класи);

w_c – це відповідний ваговий коефіцієнт для кожного із двох зазначених типів компонентів;

Склад групи компонентів у моделях РАП відповідає принципу схожості та варіативності компонентів у такому загальному понятті сучасної програмної інженерії як лінійки програмних продуктів (ЛПП), як це запропоновано, наприклад, у [158]. Для ЛПП визначаються коефіцієнти схожості структури (*Structure Similarity Coefficient – SCC*) та її варіативності (*Structure Similarity Coefficient – SVC*), що може бути аналогічним чином використано для розрахунку вагових коефіцієнтів для окремих моделей РАП, які також утворюють лінійку архітектурних примітивів (ЛАП), а саме

$$SSC = \frac{\#(CS)}{\#(CS) + \#(CV)}, \tag{3.13}$$

де #(CS) – це кількість спільних компонентів (similar components) ЛАП (компоненти типу ООП-бази); #(CV) – це кількість варіативних компонентів (variable components) ЛАП (компоненти типу ПООТ-надбудови);

$$SVC = \frac{\#(CV)}{\#(CS) + \#(CV)}, \tag{3.14}$$

де #(CS) – це кількість спільних компонентів (similar components) ЛАП (компоненти ООП-бази); #(CV) – це кількість варіативних компонентів (variable components) ЛАП (компоненти типу ПООТ-надбудови).

Отже, відповідно до виразів (3.13) – (3.14), для запропонованих моделей РАП (див. рис. 2.6 – 2.8) коефіцієнт схожості компонентів: $SSC = \frac{9}{9+13}$ буде дорівнювати 0.4, а коефіцієнт варіативності ЛАП: $SVC = \frac{13}{9+13}$ дорівнює 0.6.

Таким чином, ваговий коефіцієнт для ООП - компонентів базового типу в РАП дорівнює w_1 = 0.4, а ваговий коефіцієнт для компонентів ПООТ-типу надбудови для РАП дорівнює w_2 = 0.6.

Відповідно до виразу (3.14) загальна складність групи компонентів має бути визначена за формулою

$$ComponentComplexity = 0.6 \cdot \#(POOT) + 0.4 \cdot \#(OOP), \tag{3.15}$$

де *#(POOT)* – це кількість компонентів варіативного типу ПООТ-надбудови;
#(OOP) – це кількість компонентів схожого типу ООП-бази.

Відповідно до [123], загальна складність групи конекторів в РАП розраховується наступним чином:

$$ConnectorComplexity = \sum_{k=1}^{3} w_k \cdot \#(Q)_k; \ \sum_{k=1}^{3} w_k = 1, \tag{3.16}$$

де $\#(Q)_k$ – це загальна кількість конекторів. Індекс k – позначає приналежність конекторів до одного із трьох типів: конектори із бінарною взаємодією (binary connector interaction), конектори із множинною взаємодією (multiple connector interaction), конектори із умовною взаємодією (condition connector interaction);
w_k – це відповідний ваговий коефіцієнт для кожного із трьох зазначених типів конекторів;

Тоді для портів вираз розрахунку їх загальної складності [123] подається як:

$$PortComplexity = \sum_{r=1}^{2} w_r \cdot \#(P)_r; \ \sum_{r=1}^{2} w_r = 1, \qquad (3.17)$$

де $\#(P)_r$ – це кількість портів, індекс r – позначує приналежність відповідного порту до одного із двох типів: *простий порт* (*single port*), *порт із умовою* (*condition port*);
w_r – це відповідний ваговий коефіцієнт для кожного із двох зазначених типів портів.

Для виразів (3.16) – (3.17) доволі складно визначити критерій призначення вагових коефіцієнтів, оскілки конектори і порти є лише концептуальною абстракцією опису ПООТ за допомогою ACME ADL, для якої не передбачена їх конкретна технологічна реалізація. Тому оцінена може бути тільки міра переваги одного типу таких елементів над іншими. Для цього пропонується скористатися експертною процедурою попарного порівняння вагових коефіцієнтів за методом аналізу ієрархій (MAI) [155].

Для конекторів призначимо наступні вагові коефіцієнти:

- бінарний конектор (Binary Connector) вага $w_1=1$;
- мультиконектор (Multi Connector) вага $w_2=2$;
- конектор с умовою (Condition Connector) вага $w_3=3$.

Далі проведемо попарне порівняння призначених вагових коефіцієнтів

(див. Табл. 3.1 і Табл. 3.2), а також аналіз отриманих результатів за критеріями: індексу узгодженості (ІУ) та відношення узгодженості (ВУ) [155]:

$$IУ = \frac{(\lambda_{\max} - n)}{n-1}, \tag{3.18}$$

де $\lambda_{\max}$ – узгодженість, $\lambda_{\max} \geq n$;

n – кількість об’єктів, що порівнюються;

$$BУ = \frac{IУ}{BI}, \tag{3.19}$$

де BI – випадковий індекс (береться його табличне значення, див. [155]),

значення $BУ \leq 0{,}10$ – оптимальне.

Таблиця 3.1.

Попарне порівняння вагових коефіцієнтів конекторів

Тип конектора	Condition Connector	Multi Connector	Binary Connector
Condition Connector	1	2	3
Multi Connector	1/2	1	3/2
Binary Connector	1/3	2/3	1

Таким чином, згідно виразів (3.18) та (3.19) маємо: $IУ = 0$ та $BУ = 0$.

В результаті використання MAI для конекторів у запропонованих моделях РАП були отримані наступні значення вагових коефіцієнтів:

- бінарний конектор (Binary Connector): вага $w_1 = 0.18$;
- мультиконектор (Multi Connector): вага $w_2 = 0.27$;
- конектор с умовою (Condition Connector): вага $w_3 = 0.55$.

Таблиця 3.2.

Розрахунок вагових коефіцієнтів конекторів різних типів

Тип конектора	Condition Connector	Multi Connector	Binary Connector	Сума по строчці	Нормований вектор пріоритетів
Condition Connector	1,00	2,00	3,00	6,00	0,55
Multi Connector	0,50	1,00	1,50	3,00	0,27
Binary Connector	0,33	0,66	1,00	1,99	0,18
			Сума	10,99	1
$\lambda_{max} = 3$	ІУ = 0	ВІ = 0,58	ВУ = 0		

Таким чином, вираз (3.16) приймає наступний вид:

$$ConnectorComplexity = 0.18 \cdot \#(BinC) + 0.27 \cdot \#(MultC) + 0.55 \cdot \#(CondC), \quad (3.20)$$

де *#(BinC)* – це кількість конекторів типу бінарної взаємодії;

#(MultC) – це кількість конекторів типу множинної взаємодії;

#(CondC) – це кількість конекторів типу умовної взаємодії.

Аналогічним чином розраховуються значення вагових коефіцієнтів для портів у РАП. Призначимо наступні вагові коефіцієнти:

- простий порт (SinglePort): вага $w_1 = 1$;
- порт з умовою (ConditionPort): вага $w_2 = 3$;

Далі проведемо попарне порівняння призначених вагових коефіцієнтів (див. Табл. 3.3 – 3.4) .

Таблиця 3.3.

Попарне порівняння вагових коефіцієнтів для портів різних типів

Тип порту	Condition Port	Single Port
Condition Port	1	3
Single Port	1/3	1

Таблиця 3.4.

Розрахунок вагових коефіцієнтів для портів різних типів

Тип порту	Condition Port	Single Port	Сума по строчці	Нормований вектор пріоритетів
Condition Port	1,00	3,00	4,00	0,75
Single Port	0,33	1,00	1,33	0,25
		Сума	5,33	1

При цьому для порівняння лише двох елементів немає потреби розраховувати значення виразів (3.18 – 3.19) і таким чином отримуємо наступні значення вагових коефіцієнтів для конекторів:

– простий порт (SinglePort): вага w_1=0.25;

– порт з умовою (ConditionPort): вага w_2 = 0.75;

Тоді вираз (3.17) остаточно приймає наступний вигляд

$$PortComplexity = 0.25 \cdot \#(SinP) + 0.75 \cdot \#(CondP), \quad (3.21)$$

де *#(SinP)* – це кількість портів бінарного типу;

#(CondP) – це кількість портів умовного типу.

На основі виразів (3.15), (3.20) та (3.21) визначення загальної складності розширеного архітектурного примітива (Extended Architectural Primitive Complexity – EAPC), може бути подана як:

$$EAPC = ComponentComplexity + ConnectorComplexity + PortComplexity, \quad (3.22)$$

де *ComponentComplexity*, *ConnectorComplexity*, *PortComplexity* – це показники, що були визначені вище.

Таким чином, згідно з виразом (3.22), були отримані кількісні показники складності усіх РАП в межах лінійки архітектурних примітивів (ЛАП) для всіх

трьох досліджуваних ПООТ: АРП, ФОП та КОП. Результати розрахунків зведені у Табл. 3.5.

Таблиця 3.5.

Кількісні показники складності РАП для ПООТ

ПООТ	Складність компоненту Component Complexity	Складність конектора Connector Complexity	Складність порту Port Complexity	EAPC
АОП	4.8	1	4.3	10,10
ФОП	3.6	1	3.9	8,50
КОП	2.8	0.7	4.1	7,60

Ці результати показують, що найбільш складну архітектурну модель має технологія АОП, що відповідно потребує найбільше питомих витрат на її реалізацію, а найпростіша модель РАП належить технології КОП. Таким чином, наявні ПООТ можуть бути ранжовані наступним чином щодо порядку збільшення їх питомих витрат: КОП, ФОП, АОП.

Загальні витрати на використання ПООТ для модифікації структури УПС ($Cost^{POOT}$) визначаються також з використанням архітектурно-центрованого підходу, згідно із яким, нова ПООТ-модифікація УПС, має бути описана у термінах ALD та обчислена з урахуванням відповідних питомих затрат РАП:

$$Cost^{POOT} = EAPC^{POOT} \cdot AC^{POOT}, \qquad (3.23)$$

де $EAPC^{POOT}$ – це питома складність моделі РАП для відповідної ПООТ;

AC^{POOT} – це архітектурна складність (*Architectural Complexity – AC*) програмного рішення для модифікації УПС з використанням відповідної ПООТ, що обчислюється за тим же принципом що і складність РАП.

Отриманий кількісний показник складності (витратності) застосування окремих ПООТ для модифікації УПС з метою усунення негативних наслідків НФ в

подальшому використовується для комплексної оцінки ефективності використання ПООТ, опис процедури якої наведено у наступному параграфі цього розділу.

3.4. Процедура комплексної оцінки ефективності використання ПООТ із застосуванням методів нечіткої логіки

Вирішення *Задачі 4* в рамках запропонованого методологічного знання-орієнтованого підходу, який представлено у п. 2.1 попереднього розділу, передбачає побудову інформаційного простору $П_1$ (див. вираз (3.3),), а саме: простору для визначення оцінок ефективності використання ПООТ. Це передбачає поєднання двох різнорідних за природою параметрів: витрат на ПООТ-модифікацію УПС (див. вираз (3.23)) та ступеня присутності наскрізної функціональності (НФ) (див. вираз (2.22). З огляду на це, дана задача слабко піддається формалізації і тому для оцінки коефіцієнта ефективності використання ПООТ, що визначений виразом (2.5) (див. п.п. 2.1 попереднього розділу), пропонується використати підхід на основі застосування нечіткої логіки (fuzzy logic approach), що дозволяє коректно оперувати різнорідними показниками [147].

Загальна процедура комплексної оцінки ефективності використання ПООТ на основі метода фазифікації / дефазифікації Мамдані [159] показана на рис. 3.4. Зазначена процедура передбачає наявність початкового функціонального блоку А1, для формування множини (P) бази правил нечітких продукцій, вхідні дані: *вагові коефіцієнти визначеності* $F_i \in [0;1]$ для усіх правил $F_i = 1$, де i – загальна кількість правил нечітких продукцій; у *вербальний формі правила* записуються як вираз «якщо *А* ЛО *Б* то *В*» (що представляє собою елемент контролю, стрілка, що входить до блоку зверху – позначена на схемі як шаблон правил нечітких продукцій), А, Б – це показники *вхідних змінних*, В – показник *вихідної змінної*, ЛО – логічна умова. Правила нечітких продукцій формуються експертом (що представляється на схемі як механізм – стрілка що входить в блок знизу).

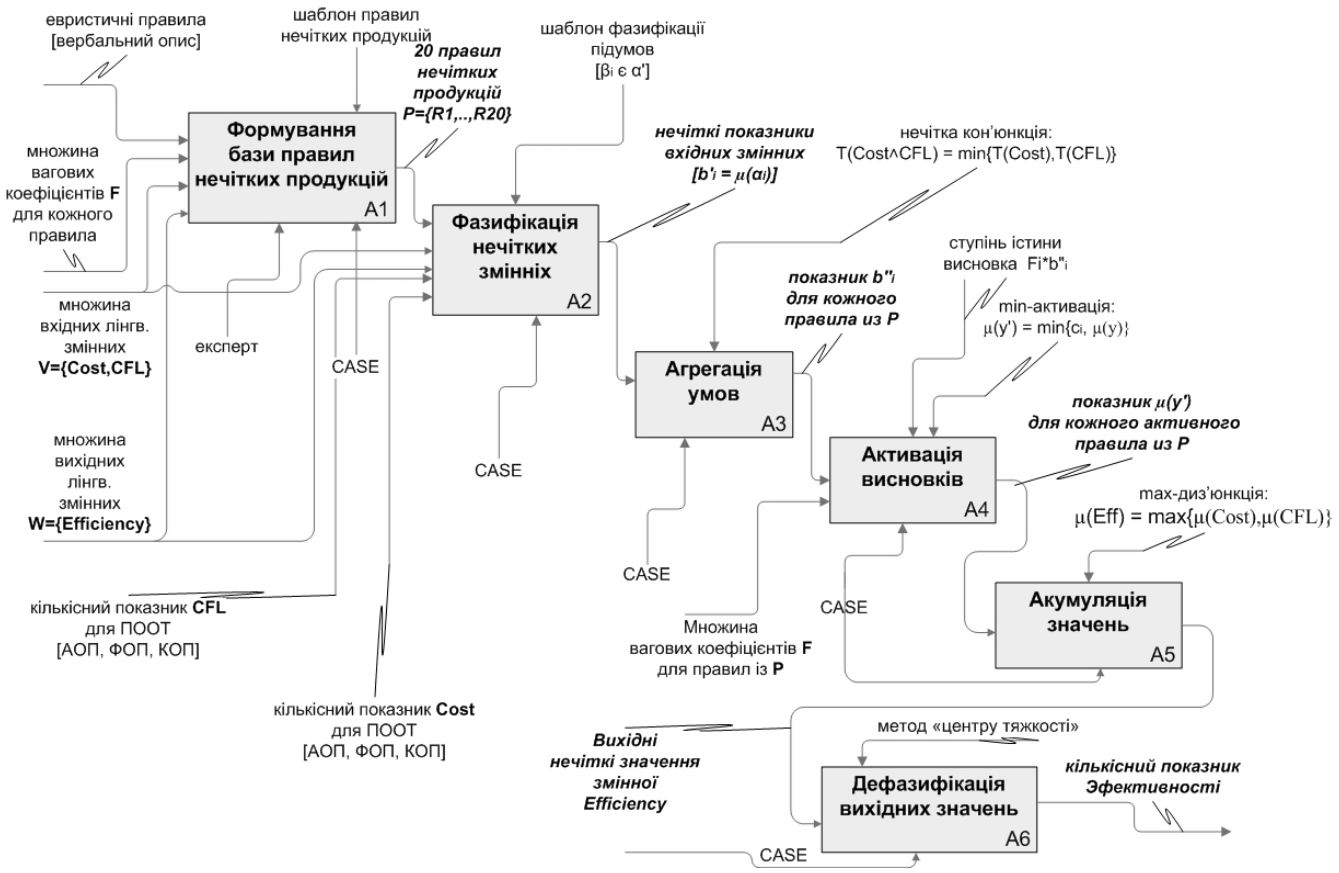


Рисунок 3.4 – Процедура оцінки ефективності використання ПООТ

1. Якщо витрати на реалізацію ПООТ-модифікації низькі і ступінь зниження НФ нульовий, то використання ПООТ є неефективним.
2. Якщо витрати на реалізацію ПООТ-модифікації середні і ступінь зниження НФ нульовий, то використання ПООТ є неефективним.
3. Якщо витрати на реалізацію ПООТ-модифікації високі і ступінь зниження НФ нульовий, то використання ПООТ є неефективним.
4. Якщо витрати на реалізацію ПООТ-модифікації дуже високі і ступінь зниження НФ нульовий, то використання ПООТ є неефективним.
5. Якщо витрати на реалізацію ПООТ-модифікації низькі і ступінь зниження НФ малий, то використання ПООТ є малоефективним.
6. Якщо витрати на реалізацію ПООТ-модифікації середні і ступінь зниження НФ малий, то використання ПООТ є малоефективним.

7. Якщо витрати на реалізацію ПООТ-модифікації високі і ступінь зниження НФ малий, то використання ПООТ є неефективним.
8. Якщо витрати на реалізацію ПООТ-модифікації дуже високі і ступінь зниження НФ малий , то використання ПООТ є неефективним.
9. Якщо витрати на реалізацію ПООТ-модифікації низькі і ступінь зниження НФ середній, то використання ПООТ є досить ефективним.
10. Якщо витрати на реалізацію ПООТ-модифікації середні і ступінь зниження НФ середній, то використання ПООТ є малоефективним.
11. Якщо витрати на реалізацію ПООТ-модифікації високі і ступінь зниження НФ середній, то використання ПООТ є малоефективним.
12. Якщо витрати на реалізацію ПООТ-модифікації дуже високі і ступінь зниження НФ середній , то використання ПООТ є неефективним.
13. Якщо витрати на реалізацію ПООТ-модифікації низькі і ступінь зниження НФ високий, то використання ПООТ є дуже ефективним.
14. Якщо витрати на реалізацію ПООТ-модифікації середні і ступінь зниження НФ високий, то використання ПООТ є ефективним.
15. Якщо витрати на реалізацію ПООТ-модифікації високі і ступінь зниження НФ високий, то використання ПООТ є малоефективним.
16. Якщо витрати на реалізацію ПООТ-модифікації дуже високі і ступінь зниження НФ високий, то використання ПООТ є малоефективним.
17. Якщо витрати на реалізацію ПООТ-модифікації низькі і ступінь зниження НФ дуже високий, то використання ПООТ є дуже ефективним.
18. Якщо витрати на реалізацію ПООТ-модифікації середні і ступінь зниження НФ дуже високий, то використання ПООТ є дуже ефективним.
19. Якщо витрати на реалізацію ПООТ-модифікації високі і ступінь зниження НФ дуже високий, то використання ПООТ є ефективним.
20. Якщо витрати на реалізацію ПООТ-модифікації дуже високі і ступінь зниження НФ дуже високий, то використання ПООТ є малоефективним.

Таблиця 3.6.

Загальновизначені скорочення для значень основних термів лінгвістичних змінних у системах нечіткого виводу

Символьне позначення	Англомовна нотація	Україномовна нотація
NB	Negative Big	Негативне велике
NM	Negative Middle	Негативне середнє
NS	Negative Small	Негативне мале
ZN	Zero Negative	Негативне близьке до нуля
Z	Zero	Нуль
ZP	Zero Positive	Позитивне близьке до нуля
PS	Positive Small	Позитивне мале
PM	Positive Middle	Позитивне середнє
PB	Positive Big	Позитивне велике
PH	Positive Huge	Позитивне дуже велике

Для скорочення запису правил нечітких продукцій використовуються позначення, які зведені у Табл. 3.6. У скороченій формі правило складається із відповідних ідентифікаторів вхідних і вихідних змінних: $ß_1$, $ß_2$, ω_1, символьних позначень (C), логічного оператору (ЛО) та записується у вигляді виразу:
«якщо β_1 є **С** ЛО β_2 є **С** то β_3 є **С**» (що також представляє собою контроль – позначене на схемі як шаблон правил нечітких продукцій).
Тоді правила нечітких продукцій приймають більш коротку форму запису
якщо $\beta_1 \in C$ ЛО $\beta_2 \in C$ то $\omega_1 \in C$.

1. ПРАВИЛО_1: Якщо β_1 є **PS** і β_2 є **Z**, то ω_1 є **Z**, (F_1);
2. ПРАВИЛО_2: Якщо β_1 є **PM** і β_2 є **Z**, то ω_1 є **Z**, (F_2);
3. ПРАВИЛО_3: Якщо β_1 є **PB** і β_2 є **Z**, то ω_1 є **Z**, (F_3);
4. ПРАВИЛО_4: Якщо β_1 є **PH** і β_2 є **Z**, то ω_1 є **Z**, (F_4);
5. ПРАВИЛО_5: Якщо β_1 є **PS** і β_2 є **PS**, то ω_1 є **PS**, (F_5);

6. ПРАВИЛО_6: Якщо β_1 є **PM** і β_2 є **PS**, то ω_1 є **PS**, (F_6);
7. ПРАВИЛО_7: Якщо β_1 є **PB** і β_2 є **PS**, то ω_1 є **Z**, (F_7);
8. ПРАВИЛО_8: Якщо β_1 є **PH** і β_2 є **PS**, то ω_1 є **Z**, (F_8);
9. ПРАВИЛО_9: Якщо β_1 є **PS** і β_2 є **PM**, то ω_1 є **PM**, (F_9);
10. ПРАВИЛО_10: Якщо β_1 є **PM** і β_2 є **PM**, то ω_1 є **PS**, (F_{10});
11. ПРАВИЛО_11: Якщо β_1 є **PB** і β_2 є **PM**, то ω_1 є **PS**, (F_{11});
12. ПРАВИЛО_12: Якщо β_1 є **PH** і β_2 є **PM**, то ω_1 є **Z**, (F_{12});
13. ПРАВИЛО_13: Якщо β_1 є **PS** і β_2 є **PB**, то ω_1 є **PH**, (F_{13});
14. ПРАВИЛО_14: Якщо β_1 є **PM** і β_2 є **PB**, то ω_1 є **PB**, (F_{14});
15. ПРАВИЛО_15: Якщо β_1 є **PB** і β_2 є **PB**, то ω_1 є **PS**, (F_{15});
16. ПРАВИЛО_16: Якщо β_1 є **PH** і β_2 є **PB**, то ω_1 є **PS**, (F_{16});
17. ПРАВИЛО_17: Якщо β_1 є **PS** і β_2 є **PH**, то ω_1 є **PH**, (F_{17});
18. ПРАВИЛО_18: Якщо β_1 є **PM** і β_2 є **PH**, то ω_1 є **PH**, (F_{18});
19. ПРАВИЛО_19: Якщо β_1 є **PB** і β_2 є **PH**, то ω_1 є **PB**, (F_{19});
20. ПРАВИЛО_20: Якщо β_1 є **PH** і β_2 є **PH**, то ω_1 є **PS**, (F_{20});

Отримані таким чином 20 правил нечітких продукцій представляють собою вихідні дані функціонального блоку A1 на рис. 3.4.

Наступний блок A2 передбачає фазифікацію лінгвістичних змінних [121] де вхідними даними є лінгвістичні змінні: «Витрати» (*Cost – C*), «Ступінь присутності НФ» (*CFL*, див. п.п. 2.5), «Ефективність» (Effectiveness – Eff), база правил нечітких продукцій, відповідно до яких буде проводитись фазифікація змінних. Вихідними даними є значення функції приналежності $b'_i = \mu(a_i)$, $i \in [1;20]$, до термів для кожної змінної відповідно до правил нечітких продукцій, де a_i – це кількісне значення, що належить до універсуму U лінгвістичної змінної.

Вхідна лінгвістична змінна (графічна інтерпретація функції приналежності якої приведена на рис. 3.5.), що відображає витрати на ПООТ-модифікацію УПС.

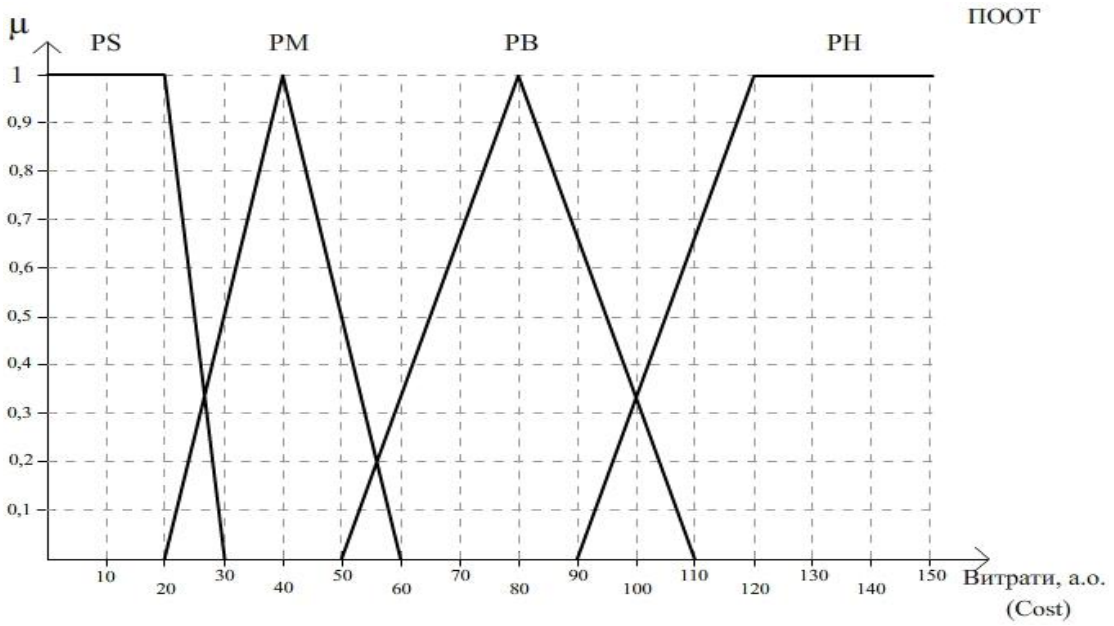


Рисунок 3.5 – Фазифікація вхідної лінгвістичної змінної «Витрати»

Вона визначається наступним чином:

X – «Витрати» («Cost»);

T(X) – {*низькі, середні, високі, дуже високі*};

T(X) у символьному вигляді, {PS, PM, PB, PH}, (див. Табл. 3.6);

U – множина значень, що визначаються у архітектурних одиницях, U є [1; ∞);

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функції приналежності: трапецієподібна та трикутна.

Вхідна лінгвістична змінна, що відображає ступінь зниження наскрізної функціональності визначається як:

X – «Ступінь присутності НФ» («CFL»);

T(X) – {*нульова*, *мала*, *середня*, *висока, дуже висока*};

T(X) у символьному вигляді, {Z, PS, PM, PB, PH}, (див. Табл. 3.6);

U – множина значень коефіцієнта зниження НФ, U є [0; 1];

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функції приналежності: трапецієподібна та трикутна.

Графічна інтерпретація функції приналежності приведена на рис. 3.6. Треба зазначити, що відповідно до виразу (2.22), із наближенням до 1, ця змінна приймає значення «нульова».

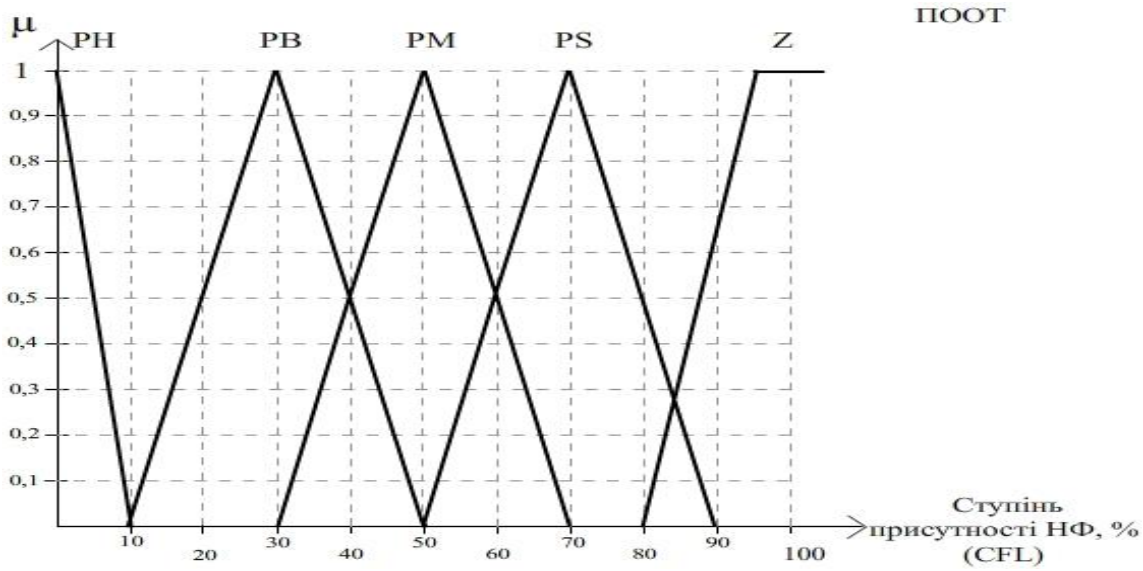


Рисунок 3.6 – Фазифікація вхідної лінгвістичної змінної «CFL»

Вихідна лінгвістична змінна, що відображає ефективність використання ПООТ визначається як:

X – «Ефективність» («Eff»);

T(X) – {*не ефективно, слабо ефективно, досить ефективно, ефективно, дуже ефективно*};

T(X) у символьному вигляді, {Z, PS, PM, PB, PH}, (див. Табл. 3.6);

U – множина значень коефіцієнта ефективності, U є [0; 1];

G – синтаксичне правило, що породжує терми множини *T(X)*,

M – функції приналежності: трапецієподібна та трикутна.

Графічна інтерпретація функції приналежності приведена на рис. 3.7.

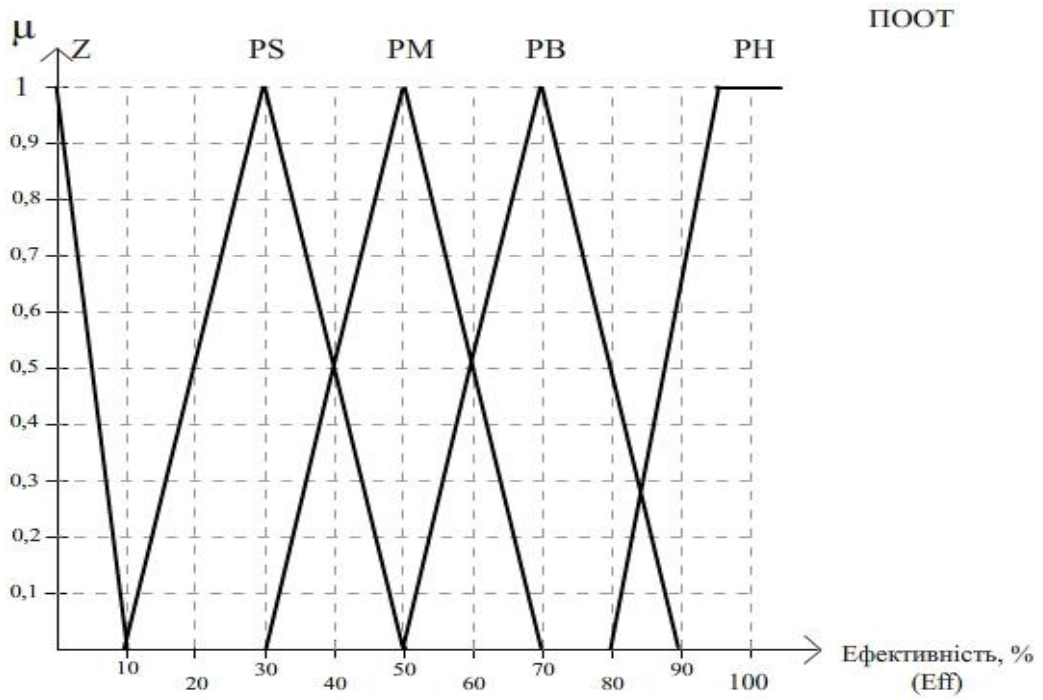


Рисунок 3.7 – Фазифікація вхідної лінгвістичної змінної «Ефективність»

Наступна функція блоку А3 передбачає агрегацію підумов для всіх правил із множини *P* (див. наприклад у [159]). Вхідними даними для цього блоку є кількісні показники витрат на реалізацію ПООТ-модифікації УПС, що відповідає множині *U* лінгвістичної змінної «Витрати»; кількісні показники ступеня зниження НФ, що відповідає множині *U* лінгвістичної змінної «CFL»; та відповідні кількісні значення функцій приналежності зазначених змінних, отриманих відповідно до графіків (див. рис. 3.5 – 3.7), що є результатом попереднього блоку. Контроль для даної функції є правило нечіткої кон’юнкції.

$$T(Cost \wedge CFL) = \min\{T(Cost), T(CFL)\}, \tag{3.24}$$

де *Cost,CFL* – це відповідні кількісні показники функцій приналежності відповідних термів вхідних змінних.

Вихідними даними агрегації підумов є значення b_i'' – результат виразу (3.24). Механізмом для даної функції є CASE-засіб.

Блок А4 передбачає активізацію підумов див. наприклад у [159]), відповідно до вагових коефіцієнтів із множини *F* для кожного правила із множини *P*, та результат попереднього блоку, що є вхідними даними для цього блоку. Контролем для даної функції є правило min-активації.

$$\mu(y') = \min\{c_i, \mu(y)\}, \tag{3.25}$$

де c_i – це алгебраїчний добуток вагового коефіцієнту *F*, для відповідного правила $r_i \in P$; $\mu(y)$ – це значення функції приналежності терму, отримане як результат попередньої активності.

Вихідними даними цієї активності є значення $\mu(y')$ – результат виразу (3.25). Механізмом для даної активності є CASE-засіб.

У блоці А5 передбачена акумуляції висновків, тобто пошук функції приналежності вихідної лінгвістичної змінної, управлінням в цьому випадку є правило max-диз'юнкції (див., напр. у [159]). Воно подається на вхід блоку А6 для дефазифікації, вихідними даними є кількісне значення коефіцієнту ефективності використання ПООТ, яка проводиться за методом «центру тяжіння» наприклад див. у [159].

Таким чином, запропонована процедура дозволяє отримати кількісну оцінку коефіцієнта ефективності використання ПООТ у залежності як від типу УПС, так і від витрат, які є пов'язаними з її структурною модифікацією із застосуванням відповідної ПООТ.

3.5. Загальна схема прикладної інформаційної технології для реалізації запропонованого підходу

Представлені вище моделі, методи та процедури, що забезпечують вирішення *Задач 1 – 4*, які були сформульовані у п. 2.1 попереднього розділу,

мають бути інтегровані в єдину прикладну інформаційну технологію (ІТ), концептуальна схема якої в нотації IDEF0 [120] представлена на рис. 3.8. Запропонована процедура складається із дев'яти функціональних блоків (ФБ), відповідно до правил нотації IDEF0 (вони вже були детально розглянуті при поданні опису схеми на рис. 2.5) кожний такий ФБ *A1, A2 ....A9* має чотири типи інтерфейсів (”Вхід”, ”Вихід”, ”Управління”, ”Механізм реалізації”). Це дозволяє відобразити зв'язок усіх ресурсів: вхідних даних, моделей, методів та алгоритмів, а також експертів та інструментальних CASE-засобів, які є необхідними для автоматизації процесу отримання оцінки ефективності використання ПООТ.

Для початкової оцінки стану УПС інформаційна технологія (див. рис. 3.8) передбачає виконання функцій, пов'язаних із оцінкою стану функціональних вимог (ФВ), що є основним джерелом постійних модифікацій УПС (див. рис. 1.4, активність 1 та 2) та початкової об'єктно-орієнтованої структурної складності УПС. оцінки основних показників ФВ (див. п. 2.3 та п. 3.2).

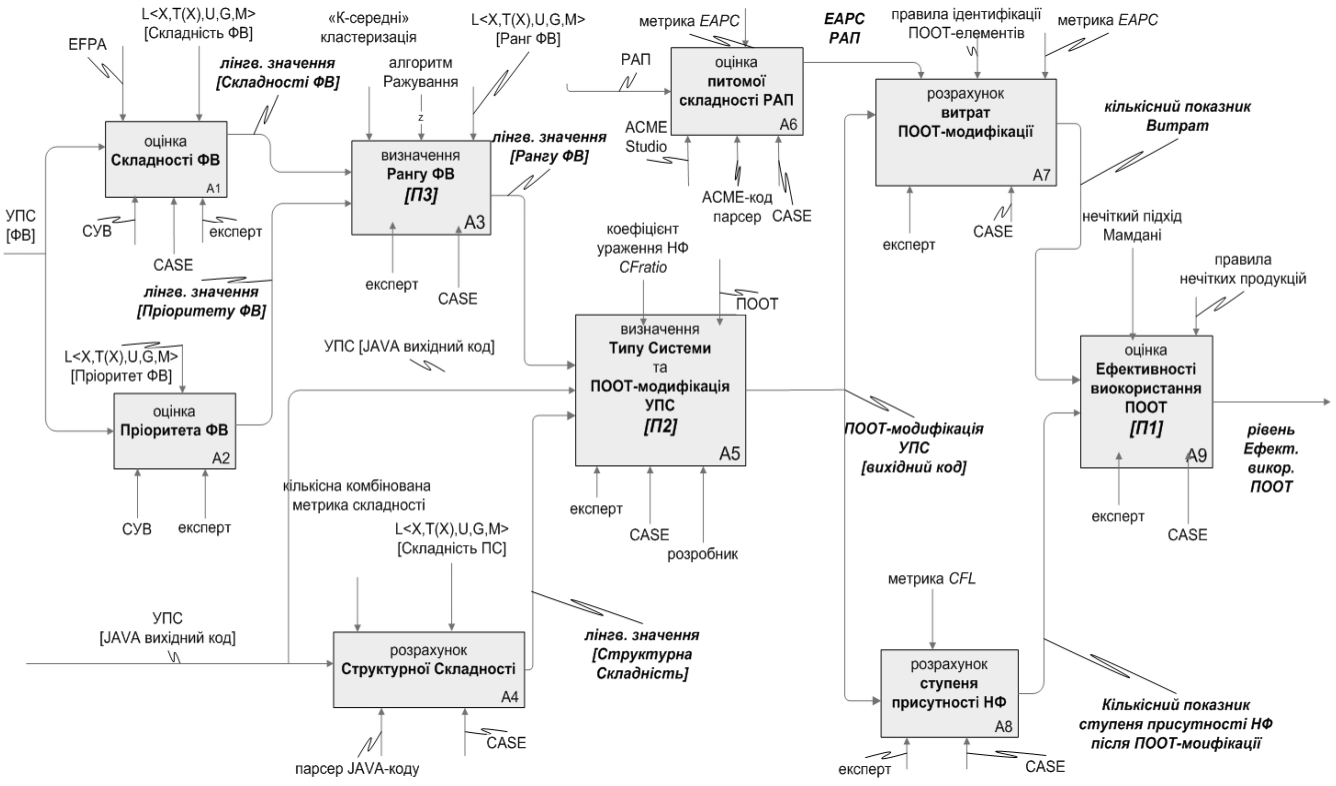


Рисунок 3.8 – Процедура оцінки кінцевої ефективності використання ПООТ

Визначенню стану вимог відповідає три функціональних блоки, що відображають:

– A1: «Оцінка пріоритету ФВ». Для оцінки пріоритету, ФВ мають бути описані у відповідній системі управління вимогами (СУВ) та повинен бути залучений експерт. Ці два елементи в термінах IDEF0 є для цього функціонального блока механізмами. Елементом управління в цьому випадку служить формалізація опису пріоритету у форму лінгвістичної змінної «Пріоритет ФВ». Результатом роботи функціонального блоку є значення із множини термів лінгвістичної змінної «Пріоритет ФВ» для кожної вимоги;

– A2: «Оцінка складності ФВ». Для оцінки складності, опис ФВ також береться із СУВ, залучення експерта дозволяє провести відповідні розрахунки із використанням CASE-засобу. Ці елементи в термінах IDEF0 є для даного функціонального блока механізмами. Елементом управління в цьому випадку служить метод ранніх функціональних балів (EFPA), який регламентує процес розрахунку кількісного показника складності кожної ФВ, що виражається в функціональних балах. Також отримані кількісні оцінки складності ФВ, мають бути описані в термінах відповідної лінгвістичної змінної. Результатом виконання цього ФБ є значення із множини термів лінгвістичної змінної «Складність ФВ» для кожної поточної ФВ;

– A3: «Визначення Рангу ФВ». Ранг вимог – це комплексний показник стану ФВ (запропонований в п. 3.2), який визначається у підпросторі *ПЗ* (див. вираз (3.5)) у складі багатовимірного інформаційного простору *InfoSpace* (див. вираз (3.2)) у загальній алгоритмічній моделі *АМ* (див. вираз (3.1)). Підпростір *ПЗ* включає два складових вимірів: складність та пріоритет ФВ, які є для ФБ A3 вхідними даними. Визначення рангу ФВ забезпечується автоматизованою процедурою із залученням CASE-засобу, який для даного ФБ представляє механізм реалізації. Елементами управління у ФБ A3 є алгоритм кластеризації відповідного підпростору із використанням алгоритму k-середніх, а також

алгоритм визначення рангу вимог, запропонований у п. 3.2, та формальний опис кількісних показників рангу у формі лінгвістичної змінної. Результатом виконання цього ФБ є отримання значення із множини термів лінгвістичної змінної «Ранг ФВ», що є комплексною характеристикою стану відповідних вимог.

Визначення оцінки початкової структурної складності УПС (див. п. 2.2 та п. 3.2) забезпечує виконання ФБ А4, а саме:

– А4: «Розрахунок Структурної Складності». Для розрахунку складності ООП–структури УПС її вихідний програмний код (напр., написаний на мові Java) слугує вхідними даними для даного блоку. Він має бути автоматично розібраний відповідним парсером (*parsing*) Java-коду, який використовує для цього абстрактне синтаксичне дерево (abstract syntax tree) для аналізу програмного коду і є складовим компонентом CASE-засобу. Ці елементи в термінах IDEF0 є для даного ФБ є механізмами реалізації. Елементами управління для цього ФБ є колекція метрик ОО-коду (див. вирази (2.6 – (2.16)), алгоритм для розрахунку структурної складності, який запропоновано у п. 3.2, та формальний опис кількісних показників структурної складності у формі лінгвістичної змінної. Результатом виконання цього ФБ є отримання відповідного значення із множини термів лінгвістичної змінної «Складність системи» для даної УПС.

Результати оцінок стану ФВ (у блоці А3) та структурної складності УПС (у блоці А4) є вхідними даними для роботи ФБ А5 для визначення типу УПС.

– А5: «Визначення Типу Системи та ПООТ-модифікація УПС». Однозначне позиціонування УПС у підпросторі типів системи *П2* (див. вираз (3.4)) на основі вхідних даних рангу ФВ та структурної складності УПС, де врахований ступінь її ураження наскрізною функціональністю (див. п.п. 2.5), із залученням механізму позиціонування CASE-засобу та контролю обраної множини типів УПС, в рамках даного дослідження таких типів визначено дев’ять (див. п.п. 3.2). Цей функціональний блок також передбачає ПООТ-модифікацію

УПС відповідного типу, на основі програмного java-коду, де елементом контролю служать парадигми трьох ПООТ, що розглядаються в даному дослідженні.

Таким чином, описана вище сукупність функціональних блоків дозволяє проводити початкову оцінку об’єктно-орієнтованої УПС, вихідний програмний код якої написаний з використанням мови Java. Виходом цього блоку є нова ПООТ-модифікація УПС відповідного типу.

Для оцінки витрат на ПООТ-модифікацію УПС інформаційна технологія (див. рис. 3.8) передбачає виконання функцій, пов’язаних із розрахунком питомих витрат та витрат на ПООТ-модифікацію УПС (див. п.п. 2.4 та 3.3).

– А6: «Оцінка питомої складності РАП». Для даного функціонального блока відповідні архітектурних моделі для АОП, ФОП, КОП розглядаються у якості вхідних даних. Із залученням відповідних засобів (що представляють собою механізми даного блоку), таких як Acme-Studio, що є середовищем розробки для мови Acme-ADL, архітектурні моделі РАП формально представляються у вигляді вихідного мета-коду, який надалі можливо імпортувати до CASE-засобу, та провести розбір парсером Acme-коду. Елементом контролю цього блоку є сукупність метрик для розрахунку складності РАП, див. вирази (3.14) – (3.22). Результатом цієї функції є колекція кількісних показників складності розглянути РАП (див. п.п. 2.4), що відповідає даним із Табл. 3.1.

– А7: «Розрахунок витрат ПООТ–модифікації». Цей функціональний блок схожий за призначенням із попереднім блоком, проте результатом його розрахунків буде кількісний показник витрат на повну модифікацію структури УПС, із залученням відповідної ПООТ. Елементом вхідних даним для цього блоку є питома складність РАП, та програмний код, УПС, що піддалася ПООТ-модифікації. Механізмами реалізації цього блоку є CASE-засіб із залученням експертів, що відповідно до правил ідентифікації ПООТ-елементів у вихідному коді модифікованої УПС, із використанням метрик складності (3.12) – (3.20), проводять відповідні розрахунки. Результатом блоку є кількісний показник витрат

для повної ПООТ-модифікації УПС із залученням однієї із трьох ПООТ, що розглядаються у цій дисертаційній роботі.

Таким чином, така сукупність функціональних блоків А1 – А7 дає можливість отримати кількісні оцінки витрат для ПООТ-модифікації структури УПС із залучанням технологій АОП, ФОП, або КОП.

Для розрахунку ступеня зниження НФ запропонована технологія передбачає наявність відповідного ФБ А8, а саме:

– А8: «Розрахунок ступеня присутності НФ». У якості вхідних даних блок приймає вихідний код ПООТ-модифікації УПС, та із залученням експерта і використанням CASE-засобу, на основі відповідної комплексної метрики (див. вираз (2.22)) надає результат, який представляє кількісний показник ступеня присутності НФ у відповідній УПС.

Нарешті, для остаточної оцінки ефективності використання ПООТ, розроблена технологія передбачає виконання ФБ А9.

– А9: «Оцінка Ефективності використання ПООТ». Це комплексний показник, що представлений у вигляді підпростору *П1* (див. вираз (3.3)) і є складовою багатовимірного інформаційного простору (див. вираз (3.2)). Вхідними даними цього блоку є різнорідні за природою показники витрат на ПООТ-модифікацію та зниження рівня НФ, що відповідно, розрахований для заданої ПООТ-модифікації. Елементами управління у цьому ФБ є процедура фазифікації / дефазифікації за методом Мамдані (опис відповідної процедури див. у п. 3.4) та правила нечітких продукцій. Вихідні дані цього ФБ – це кількісний показник рівня ефективності використання однієї із трьох розглянутих ПООТ для визначеного типу УПС.

Таким чином, запропонована прикладна інформаційна технологія агрегує в собі всі локальні рішення відповідних задач (із списку *Задача 1 – Задача 5*, див. п. 2.1) та дає можливість, на базі кількісних та експертних оцінок, отримати

остаточну комплексну кількісну оцінку ефективності використання ПООТ для певного типу УПС.

Висновки по розділу 3

У даному розділі дисертаційної роботи:

1. Розроблена алгоритмічна модель процесу експертної оцінки ефективності ПООТ при супроводі успадкованих програмних систем (УПС).

2. Запропоновано метод визначення типу УПС із урахуванням таких чинників як структурна складність їх програмного коду, а також рівень складності та динаміки змін у функціональних вимогах користувачів таких систем.

3. Розроблені метрики та алгоритми визначення питомих витрат на проведення структурної модифікації УПС із застосуванням певної ПООТ на основі їх архітектурних моделей.

4. Запропонована процедура комплексної оцінки ефективності застосування ПООТ на основі застосування методів нечіткої логіки.

5. Представлена загальна схема прикладної інформаційної технології для автоматизації виконання всіх основних етапів запропонованого підходу.

Нові наукові результати, викладені в цьому розділі, опубліковані в роботах [123, 147, 150, 153].

РОЗДІЛ 4.
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ МОДЕЛЕЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

В цьому розділі розглянуті питання програмної реалізації інструментальних засобів для експериментальної перевірки працездатності та ефективності запропонованих моделей та процедур для визначення використання ПООТ в процесах супроводу УПС, а також проаналізовані результати відповідних програмних тестів і чисельних розрахунків.

У першому параграфі наведено опис предметної області застосування конкретного класу УПС, для яких в подальшому проводиться дослідження працездатності та ефективності запропонованого підходу.

Другий параграф присвячено висвітленню питань розробки архітектури, бази даних та основних програмних компонентів інструментального CASE-засобу для реалізації запропонованої інформаційної технології.

У третьому параграфі представлені методика проведення обчислювальних експериментів, вхідні дані та приклади виконання розрахунків окремих етапів.

Четвертий параграф містить аналіз отриманих результатів та практичні рекомендації щодо ефективного застосування існуючих ПООТ для супроводу окремих типів УПС.

4.1. Опис предметної області для задачі вибору ПООТ в процесі супроводу УПС

Не обмежуючи загального бачення запропонованого підходу до розробки та супроводу складних ПС із застосуванням ПООТ, у подальшому в цьому дослідженні розглядаються процеси супроводу інформаційно-аналітичних ПС, тобто це не є системи обробки даних у реальному часі, або системи з

автоматизації критичних технологічних процесів, які мають специфічні вимоги щодо продуктивності (performance) та надійності (reliability) їх функціонування [107]. Крім того, передбачається, що такі ПС

- є успадкованими, тобто належать до класу УПС, характерні риси яких були вже зазначені у п. 1.1 дисертації;
- ці УПС були розроблені на основі застосування парадигми ООП, наприклад, засобами мови програмування Java [9];
- в процесі супроводу таких УПС з'явилася проблема появи НФ у модулях, які реалізують бізнес-логіку їх предметної області (ПрО).

Прикладом такого класу УПС може вважатися інформаційно-аналітична ПС для автоматизації адміністрування та документообігу в міжрегіональній мережі навчальних закладів [147].

Топологія об'єктів цієї ПрО має ієрархічну багаторівневу структуру, яка представлена на рис. 4.1, в якій присутні чотири рівня організаційно-управляючих підрозділів, а саме

- *рівень **I***: на ньому знаходиться обчислювальний центр (ОЦ) і розгорнута серверна частина УПС;
- *рівень **II***: на ньому розміщені регіональні відділення, які виконують відповідну адміністративно-управляючу функцію;
- *рівень **III***: вузли цього рівня організаційно відображають районні управляння освіти, які безпосередньо відповідають за функціонування окремих навчальних закладів;
- *рівень **IV**:* на ньому знаходяться самі навчальні заклади, в яких фізично розміщені та функціонують клієнтські програмні компоненти УПС.

З іншого боку, з точки зору функціонального наповнення, ця мережева система має тільки дві групи програмних компонентів, а саме

1) серверні компоненти різного призначення (реалізація БД, рівень бізнес-логіки та ін.);

2) клієнтські компоненти, які безпосередньо імплементують візуальні інтерфейси кінцевих користувачів.

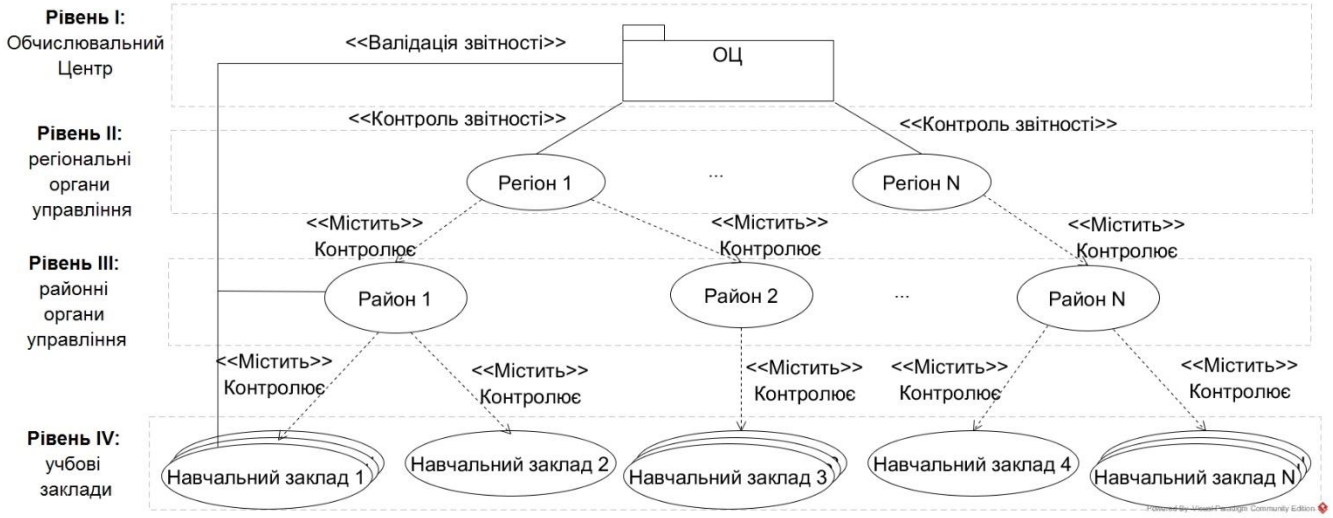


Рисунок. 4.1 – Ієрархічна структура предметної області

Таким чином, можна стверджувати, що взаємодія програмних компонентів груп (1) та (2) відповідає дворівневій еталонній архітектурі клієнт-серверних застосувань [52], діаграма розміщення якої представлена в нотації UML 2.0 (у середовищі CASE-засобу моделювання Visual Paradigm [161]) на відповідній діаграмі розміщення, рис. 4.2.

УПС фізично розміщена на двох базових типах вузлів, на яких знаходяться комп’ютери з відповідними операційними середовищами виконання окремих програмних компонентів, до яких відносяться:

– *робоча станція користувача* (user work station) представляє клієнтський вузол, на якому запущений *веб-браузер*, що на схемі позначений як компонент;
– *сервер балансування навантаження* (load-balance server) представляє серверну групу вузлів (пристроїв) на якому розгорнуте відповідне середовище виконання *сервер HAP Proxi* [162], що виконує функцію балансувальника навантаження запитів, та використовує відповідний

протокол PROXY protocol, для спрямування HTTP/HTTPS запитів, які надходять від клієнтів на різні веб-сервери;

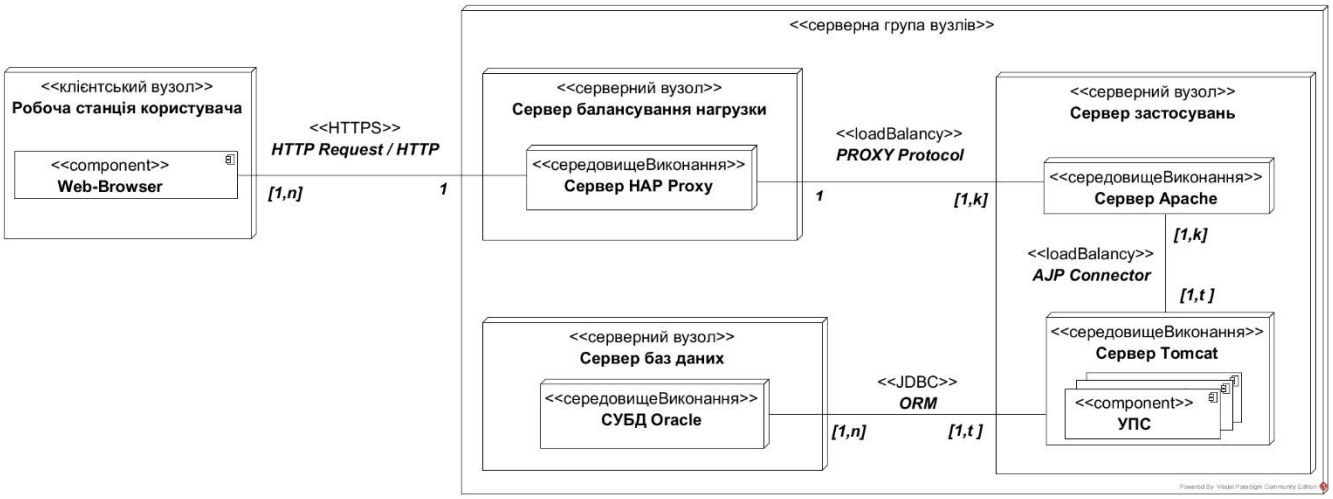


Рисунок. 4.2 – Спрощена схема розміщення основних компонентів УПС

- *сервер застосувань* (application server) входить до серверної групу вузлів (пристроїв) на якому розгорнуті два середовища виконання, а саме
  - *веб-сервер Apache* [163], який приймає відповідні переспрямовані балансувальником навантаження HTTP/HTTPS запити, та через AJP Connector (що є складовою балансування навантаження) спрямовує їх до необхідного серверу, на якому безпосередньо розгорнута УПС;
  - *сервер Tomcat* [164] середовище виконання, в якому безпосередньо розміщені для виконання складові компоненти УПС. На такому сервері одночасно розгорнуто близько 20 екземплярів УПС;
- *сервер баз даних* (database server) входить до серверної групи вузлів (пристроїв) на якому розгорнуте середовище виконання системи керування базою даних (СКБД) Oracle [165]. Відповідно цей сервер відповідає за зберігання та оброку запитів до БД. Взаємодія СУБД із УПС відбувається через встановлення JDBC-з’єднання та об’єктно-реляційного відображення даних (object-relational mapping) в Java-об’єкти.

Підсумовуючи результати аналізу топології ПрО та архітектурних особливостей розміщення окремих компонентів УПС, можна стверджувати, що система такого типу має основні характеристики, які представлені Табл. 4.1.

Таблиця 4.1.

Основні характеристики УПС, що розглядається в дослідженні

№	*Характеристика ПС*	*Вербальний опис*
1.	Фаза ЖЦ	Супровід
2.	Вид ПС	Успадкована програмна система (УПС)
3.	Тип функціональності ПС	Інформаційно-аналітична
4.	Сфера застосування	Адміністрування та автоматизація документообігу в учбових закладах
5.	Парадигма розробки	ООП
6.	Архітектура	Клієнт-серверна, 2-х рівнева
7.	Мова програмування серверної частини	Java
8	Мови програмування клієнтської частини ПС	JavaScript, HTML, CSS
9.	Основний архітектурний шаблон	MVC (Model-View-Controller) (модель-відображення-контролер)
10.	Технологія доступу до даних	об’єктно-реляційне відображення ORM (object-relational mapping)
11.	Кількість функціональних (логічних) модулів	Від 20 до 25
12.	Кількість записів в одній таблиці БД	Приблизно від ~ 10^6 до ~ 10^7
13.	Середня кількість файлів програмного коду	У серверній частині ~ 10^3; У клієнтський частині ~ 10^2
14.	Середня кількість розгорнутих екземплярів системи	Станом на вересень 2015 року ~ 10^2

УПС у даній ПрО активно експлуатуються користувачами, що призводить до великої кількості запитів на модифікацію та частого розгортання нових версій УПС, в яких реалізовані необхідні функціональні модифікації та виправлені виявлені дефекти. Приклад кількості реалізованих командою супроводу

модифікацій у такій УПС приведений в Табл. 4.2., це дані в за період с 2013 року по 2015 рік (перше півріччя).

Таблиця 4.2.

Кількість реалізованих запитів на модифікацію командою супроводу для УПС

Середня кількість змін за проміжок часу (кількість зареєстрованих запитів на модифікацію УПС)		
2013 р.	2014 р.	6 міс. 2015 р.
223	424	188

Таким чином, ПС для даної ПрО представляють собою функціонально досить складні програмні рішення із значними об’ємами даних, що мають бути оброблені з достатнім рівнем надійності в процесі їх активної експлуатації, про що свідчить велика кількість запитів на модифікацію. Такі системи представляють цінність для кінцевих користувачів, тобто повністю підходять до поняття ”успадкована ПС (УПС)”, яке було розглянуто у п. 1.1 першого розділу дисертації.

4.2. Розробка архітектури та основних програмних компонентів інструментального CASE-засобу для реалізації запропонованого підходу

Для автоматизації комплексної процедури оцінки ефективності використання ПООТ, що була запропонована у п.п. 3.5 третього розділу, був розроблений прототип відповідного CASE-засобу [166] (див. рис. 4.3).

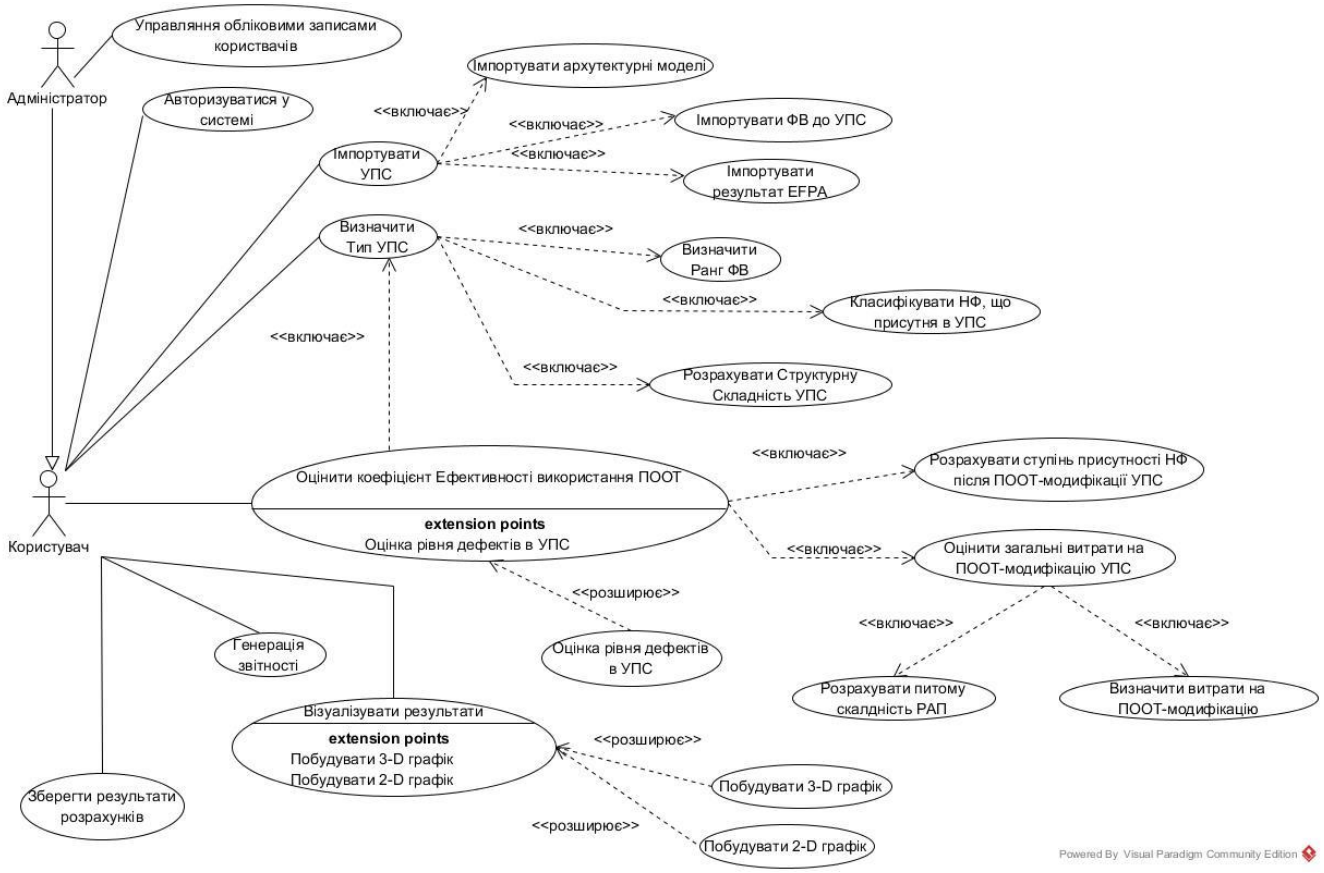


Рисунок. 4.3 – Варіанти використання розробленого CASE-засобу

Діаграма на рис. 4.3 представляє основні функціональні можливості розробленого інструментарію у вигляді UML-діаграми сценаріїв (use-case diagram) [167]. Даний CASE-засіб передбачає два типи користувачів: «Експерт» – це користувач, який здійснює основну взаємодію із інструментарієм, «Адміністратор» – це користувач, який має додаткові можливості для управління обліковими записами користувачів. Відповідно до зазначеної вище комплексної процедури (див. рис. 3.8), розроблений інструментарій передбачає основні функції:

– «Імпортувати УПС» – це функція CASE-засобу, яка передбачає отримання початкових даних (опис ФВ, результат оцінки складності ФВ, опис архітектурних моделей РАП та програмний вихідний java-код УПС) про досліджувану УПС із зовнішніх джерел у відповідному форматі;

– «Визначити тип УПС» – це функція CASE-засобу, яка передбачає можливість типізації досліджуваної УПС і включає в себе необхідні для попередньої обробки даних функції, такі як: «Визначити Ранг ФВ», «Розрахувати Структурну Складність УПС», «Класифікувати НФ, що присутня в УПС»;

– «Оцінити загальні витрати на ПООТ-модифікацію УПС» – це функція CASE-засобу, яка передбачає можливість оцінки витрат, які потрібні для модифікації УПС з використанням відповідної ПООТ. Вона включає в себе необхідні для попереднього виконання функції, такі як: «Розрахувати питому складність РАП» та «Визначити витрати на ПООТ-модифікацію»;

– «Розрахувати ступінь присутності НФ після ПООТ-модифікації УПС» – це функція CASE-засобу, яка передбачає можливість виміряти залишкову присутність НФ, після модифікації програмного коду УПС із використанням відповідної ПООТ;

– «Оцінити коефіцієнт ефективності використання ПООТ» – це центральна функція CASE-засобу, яка включає в себе попереднє виконання зазначених вище функцій: «Визначити тип УПС», «Розрахувати ступінь присутності НФ після ПООТ-модифікації УПС» та «Оцінити загальні витрати на ПООТ-модифікацію УПС», та дає можливість отримати кількісну оцінку ефективності використання ПООТ. Також ця функція розширена додатковою можливістю оцінки рівня дефектів в УПС, відповідно до п.п. 1.2, першого розділу.

Розроблений CASE-засіб представляє собою інструментальне програмне середовище, яке крім функцій обчислювального характеру має є ряд додаткових функціональних можливостей, таких як:

– «Візуалізувати результати» – це допоміжна функція, яка надає можливість візуалізації процесу розрахунку, наприклад: «Побудувати 2-D графік» (простори $П_3$ «Ранг ФВ» та $П_2$ «Тип ПС», див. п. 3.1 у третьому розділі) та «Побудувати 3-D графік» (відповідний простір $П_1$ «Ефективність використання ПООТ»);

– «Генерація звіту» – це допоміжна функція, яка надає можливість згенерувати у відповідному форматі звітність щодо процедури оцінки ефективності використання ПООТ для супроводу досліджуваної УПС;

– «Зберегти результати розрахунків» – це допоміжна функція, яка надає можливість збереження даних про зазначену процедуру у відповідній базі даних (БД).

Таким чином, розроблений CASE-засіб має основну функціональність, яка складається із п’яти основних функцій, що відповідають основним етапам процедури оцінки ПООТ, і додаткову функціональність, яка складається із трьох утилітарних функцій із маніпулювання результатами відповідних розрахунків та адміністративну функціональність, яка складається із однієї функції «Управління обліковими записами користувачів».

Відповідно до попередньо розроблених функціональних можливостей прототипу CASE-засобу була розроблена концептуальна модель даних (МД) із використанням нотації IDEF1X [120], фрагмент якої приведений на рис. 4.4 (більш докладна ця МД приведена на рис. Б.1 у додатку Б). Даний фрагмент відповідає програмному компоненту для розрахунку коефіцієнта ефективності використання ПООТ в процесі супроводу УПС (див. п.п. 3.4). Основними сутностями цього фрагменту МД є такі:

– «POOT» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про ПООТ;
– «LegacySystem» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про УПС та шлях відповідний шлях до неї у файловій системі;
– «POOTImplementation» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про модифікацію програмного коду УПС із використанням конкретної ПООТ;

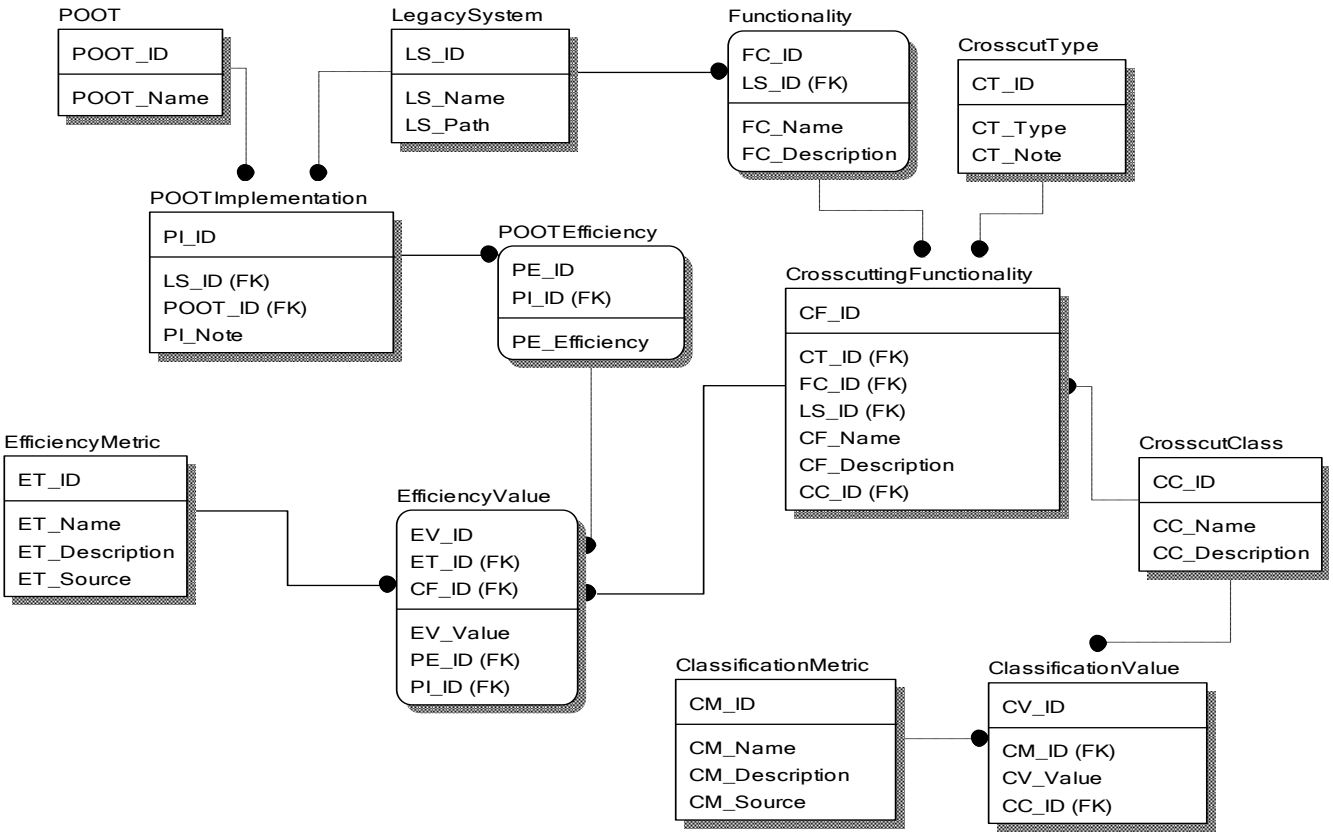


Рисунок. 4.4 – Фрагмент концептуальної моделі даних розробленого CASE-засобу

– «POOTEfficiency» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про кількісні показники витрат для ПООТ-модифікації програмного ООП-коду УПС;
– «Functionality» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про наявність НФ у конкретній УПС, що досліджується;
– «CrosscutType» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про можливі типи НФ (див. п.п. 1.2 та 2.5, у розділах 1 та 2);
– «CrosscutClass» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про можливі класи НФ (див. п.п 2.5 розділу 2);
– «ClassificationMetric» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про метрики НФ та їх одиниці вимірювання;

– «ClassificationValue» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про кількісні показники НФ, що віднесені до визначеного попередньо класу НФ;
– «CrosscuttingFunctionality» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає агреговані дані про НФ у конкретній УПС;
– «EfficiencyMetric» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані що описують метричний показник ефективності використання ПООТ, та його одиниці виміру;
– «EfficiencyValue» – це сутність, що представляє відповідну таблицю фізичного рівня та зберігає дані про кількісний показник ефективності використання ПООТ в процесі супроводу УПС.

Таким чином, МД, що відповідає програмному компоненту для оцінки ефективності використання ПООТ, складається із дванадцяти сутностей (таблиць фізичного рівня), та призначена для розрахунків кількісного показника ефективності використання ПООТ. Концептуальна МД також відображає функції розробленого прототипу CASE-засобу (рис. 4.3), які надають можливість збереження даних відповідних розрахунків та функцію генерації звітності на основі даних, які зберігаються у базі даних.

Для реалізації функціональних можливостей, опис яких наведений вище (рис. 4.3), були розроблені програмні компоненти, які схематично показані на компонентній UML-діаграмі (component diagram) [167], рис. 4.5.

Основними компонентами розробленого прототипу CASE-засобу є такі:

– «Інтегроване середовище» – основний компонент, який відповідає за обмін даними між іншими компонентами прототипу CASE-засобу. До його складу входить множина необхідних для надання інтерфейсів (required interface) для імпорту вхідних даних про УПС, а саме:

- «ILSSJParsable» – інтерфейс, який служить для реалізації функції розбору (parsing) програмного java-коду УПС, для подальших маніпуляцій із конструкціями java-коду, наприклад для розрахування структурної складності УПС;
- «IEAPParsable» – інтерфейс, який служить для реалізації функції розбору (parsing) мета-коду архітектурних моделей (із використанням мови Acme-ADL [136]) РАП, згенерованого відповідним зовнішнім засобом для проектування архітектур на основі, а саме AcmeStudio [168]. Приклад мета-коду приведений у додатку В.

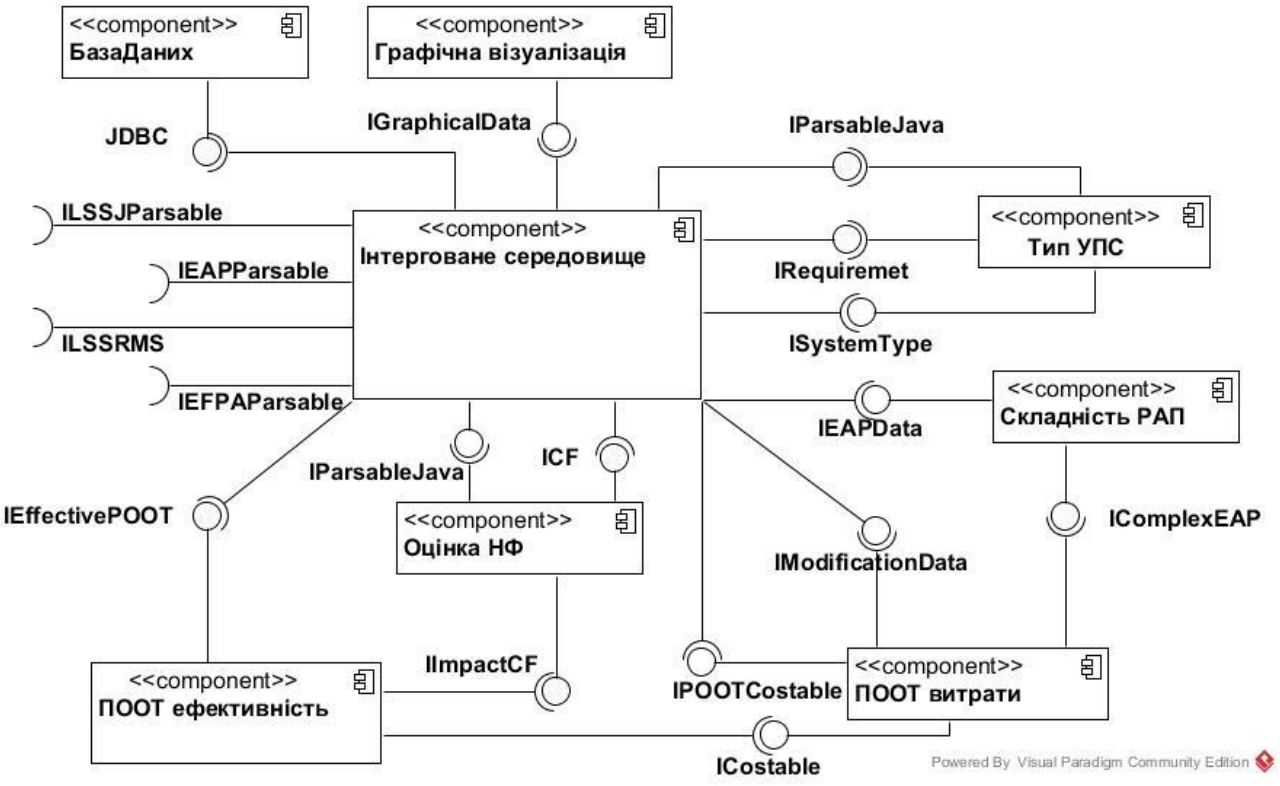


Рисунок. 4.5 – Компонентна діаграма розробленого CASE-засобу

- «ILSSRMS» – інтерфейс, який служить для реалізації функції розбору (parsing) документу, який є результатом експорту даних із СУТ IBM Rational Requisite Pro [109].

- «IFPAParsable» – інтерфейс, який служить для реалізації функції розбору (parsing) документу у форматі XML, який є результатом експорту даних із інструментального середовища моделювання функціональної складності ФВ, а саме: Function Point Modeler [169].

– «Тип УПС» – основний компонент, який відповідає за розрахунок усіх необхідних кроків для визначення типу конкретної УПС. До його складу входить множина необхідних для надання інтерфейсів (required interface) для взаємодії із компонентом «Інтегроване середовище», та отримання від нього необхідних даних, та один інтерфейс, що надається (provided interface), реалізацію якого який потребує компонент «Інтегроване середовище», а саме:

- «IParsableJava» – required interface, який дозволяє отримати підготований програмний java-код для відповідної операції розрахування структурної складності УПС;
- «IRequirement» – required interface, який дозволяє отримати у необхідному форматі функціональні вимоги, для подальшої операції визначення рангу ФВ;
- «ISystemType» – provided interface, який реалізує даний компонент, для надання даних про розрахований тип УПС.

– «Складність РАП» – компонент розробленого прототипу CASE-засобу, який виконує операцію розрахунку складності розширених архітектурних примітивів на основі опису у мета-коді їх архітектурних моделей. До його складу входить один інтерфейс, що надається (provided interface), реалізацію яких потребують компоненти «Інтегроване середовище» та «ПООТ витрати» та один необхідний для надання інтерфейс (required interface) для взаємодії із компонентом «Інтегроване середовище», а саме:

- «IEAPData» – required interface, який дозволяє отримати підготовані дані про архітектурні молі РАП, для відповідного розрахунку їх питомої складності;

- «IComplexEAP» – provided interface, який реалізує даний компонент, для надання даних про розраховану питому складність моделей РАП для компоненту «ПООТ витрати».

– «ПООТ витрати» – компонент розробленого прототипу CASE-засобу, який виконує операцію з розрахунку витрат на ПООТ-модифікації УПС. До його складу входить множина необхідних для надання інтерфейсів (required interface) для взаємодії із компонентами «Інтегроване середовище» та «Складність РАП», а також множина інтерфейсів, що надається (provided interface), реалізацію яких потребують компоненти «Інтегроване середовище» та «ПООТ ефективність», а саме:

- «IComplexEAP» required interface, який дозволяє отримати підготовані дані про питому складність відповідних РАП від компоненту «Складність РАП», для подальшого розрахунку;
- «IModificationData» – required interface, який дозволяє отримати підготовані дані про відповідні ПООТ-модифікації УПС, від компоненту «Інтегроване середовище», для подальшого розрахунку витрат на ПООТ-модифікації;
- «IPOOTCostable» – provided interface, який реалізує даний компонент, для надання даних про розраховані витрати на ПООТ-модифікації, для компоненту «Інтегроване середовище»;
- «ICostable» » – provided interface, який реалізує даний компонент, для надання даних про розраховані витрати на ПООТ-модифікації, для компоненту «ПООТ ефективність»;

– «Оцінка НФ» – компонент розробленого прототипу CASE-засобу, який виконує операції з оцінки присутності НФ у конкретній УПС. До його складу входить множина інтерфейсів, що надається (provided interface), реалізацію яких потребують компоненти «Інтегроване середовище» та «ПООТ ефективність», а

також ще один інтерфейс (required interface), який є необхідним для взаємодії із компонентами «Інтегроване середовище», а саме:

- «IParsableJava» – required interface, який дозволяє отримати підготований програмний java-код для відповідних операцій із визначення класу наскрізної функціональності в УПС, та розрахування ступеня присутності НФ після реалізації відповідних ПООТ-модифікацій;
- «ICF»– provided interface, який реалізує даний компонент, для надання даних про визначений клас та розрахований ступінь присутності НФ, для компоненту «Інтегроване середовище»;
- «IImpactCF» – provided interface, який реалізує даний компонент, для надання даних про розрахований ступінь присутності НФ, для подальших розрахунків у компоненті «ПООТ ефективність»;

– «ПООТ ефективність» – компонент розробленого прототипу CASE-засобу, який виконує операцію розрахунку ефективності використання ПООТ при супроводу УПС. До його складу входить один інтерфейс, що надається (provided interface), реалізацію якого потребує компонент «Інтегроване середовище», а також множина необхідних для надання інтерфейс (required interface) для взаємодії із компонентами «Оцінка НФ» та «ПООТ витрати», а саме:

- «ICostable» – опис цього інтерфейсу наведений вище;
- «IImpactCF» – опис цього інтерфейсу наведений вище;
- «IEffectivePOOT» – provided interface, який реалізує даний компонент, для надання даних про розраховану ефективність ПООТ, для компоненту «Інтегроване середовище»;

– «Графічна візуалізація» – компонент розробленого прототипу CASE-засобу, який виконує операцію графічного відображення даних розрахунків, які отримує компонент «Інтегроване середовище» від компонентів, які безпосередньо виконують операції із відповідних розрахунків. До його складу входить один

інтерфейс, що надається (provided interface), реалізацію якого потребує компонент «Інтегроване середовище» для обміну відповідними даними, а саме «IGraphicalData».

– «База даних» – компонент розробленого прототипу CASE-засобу, який на діаграмі компонентів (рис. 4.5) відображає зовнішню базу даних, і відповідає за зберігання даних розрахунків. Обмін даними з БД виконується з використанням інтерфейсу з’єднання з базою даних JDBC (Java Database Connectivity).

Деталізація компоненту визначення типу УПС, наведена додатку Б на рис. Б.2 та Б.3. Таким чином, розроблений прототип CASE-засобу складається із восьми основних програмних компонентів, що реалізують вісімнадцять інтерфейсів двох приведених вище типів.

На рис. 4.6 приведений фрагмент інтерфейсу користувача CASE-засобу для розрахунку рангу вимог.

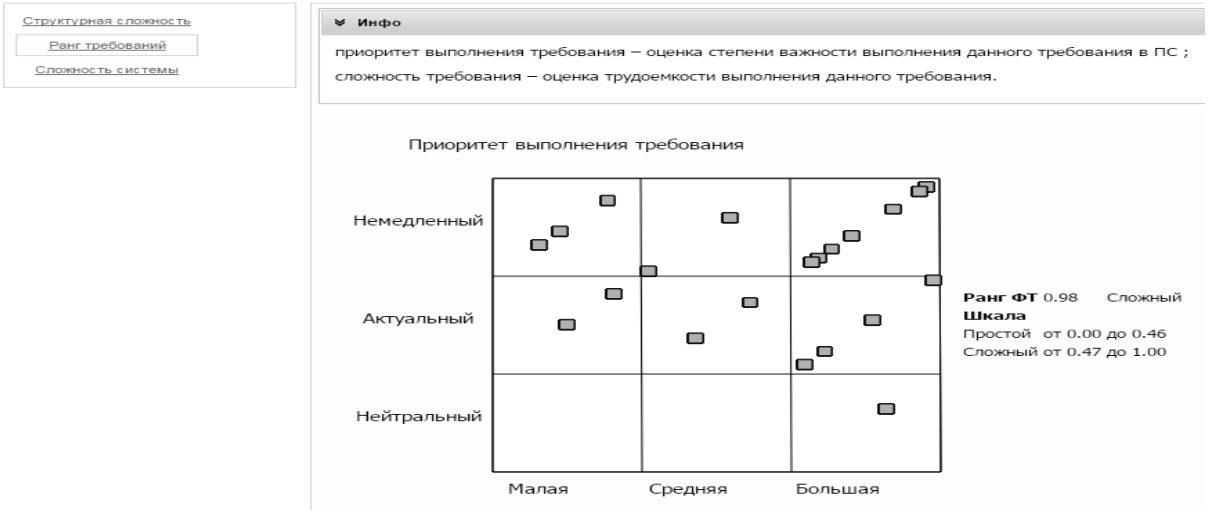
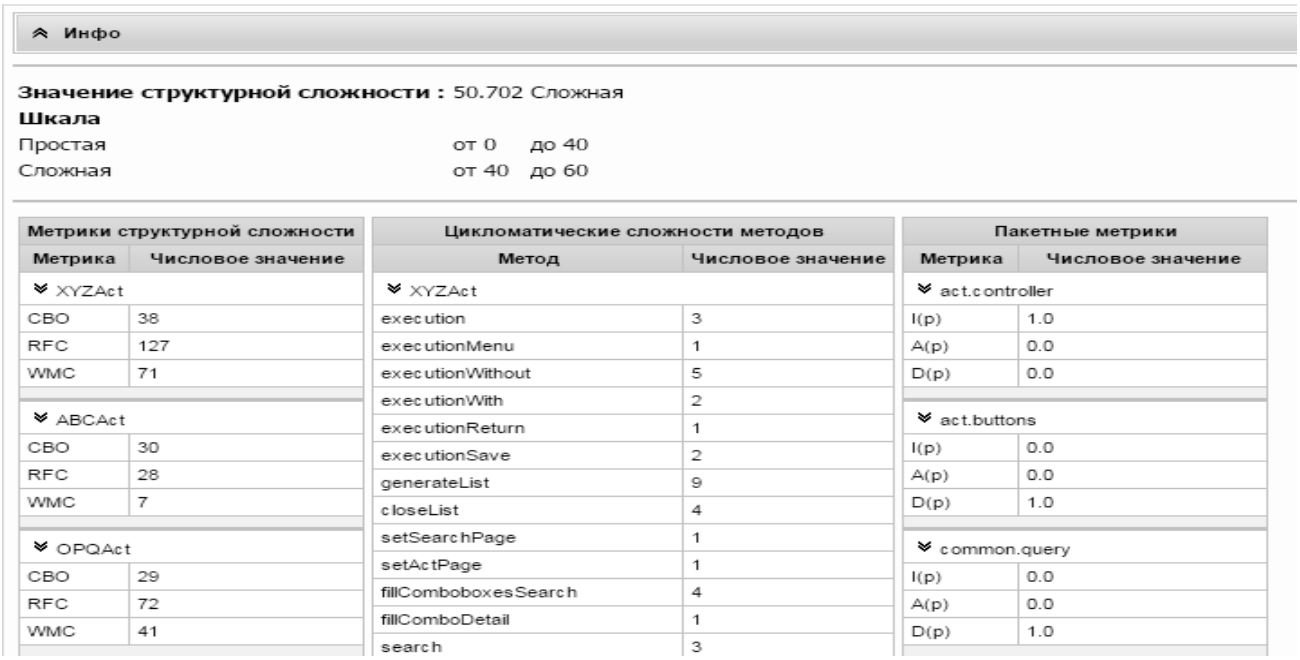


Рисунок 4.6 – Фрагмент інтерфейсу користувача «Ранг ФВ»

Він складається із двох основних частин, а саме: даних щодо кількісних показників рангу та графічної інтерпретації 2D-простору $П_2$ «Ранг ФТ».

На рис. 4.7 відображений фрагмент інтерфейсу користувача для розрахунку структурної складності УПС, який складається із таблиці кількісних показників

загальної структурної складності УПС, та деталізації даних розрахунків, що реалізовані як списки, що розгортаються при кліку на них лівою кнопкою маніпулятора миші.


Инфо

Значение структурной сложности : 50.702 Сложная

Шкала

Простая	от 0	до 40
Сложная	от 40	до 60

Метрики структурной сложности	
Метрика	**Числовое значение**
XYZAct	
CBO	38
RFC	127
WMC	71
ABCAct	
CBO	30
RFC	28
WMC	7
OPQAct	
CBO	29
RFC	72
WMC	41

Цикломатические сложности методов	
Метод	**Числовое значение**
XYZAct	
execution	3
executionMenu	1
executionWithout	5
executionWith	2
executionReturn	1
executionSave	2
generateList	9
closeList	4
setSearchPage	1
setActPage	1
fillComboboxesSearch	4
fillComboDetail	1
search	3

Пакетные метрики	
Метрика	**Числовое значение**
act.controller	
I(p)	1.0
A(p)	0.0
D(p)	0.0
act.buttons	
I(p)	0.0
A(p)	0.0
D(p)	1.0
common.query	
I(p)	0.0
A(p)	0.0
D(p)	1.0

Рисунок 4.7 – Фрагмент інтерфейсу користувача «Структурна складність УПС»

Рис. 4.8 показує розрахунок питомої складності РАП та її графічну інтерпретацію у вигляді графіку типу «Радар», а також інформаційний блок, який відображає архітектурну модель РАП для однієї із трьох досліджуваних ПООТ, його можна відкрити додатково.

На рис. 4.9 та 4.10 відповідно відображені фрагменти інтерфейсу користувача, які призначені для введення вхідних даних (та зберігання її у БД) та розрахунку коефіцієнта ефективності використання ПООТ відповідно.

На рис. 4.10 представлені результати кінцевого розрахунку коефіцієнта ефективності ПООТ K_{Eff} .

Кількісні показники ефективності представлені у табличній формі, графічна інтерпретація 3D-простору Π_1 «Ефективність використання ПООТ» показує результат розрахунку відповідно до типу УПС.

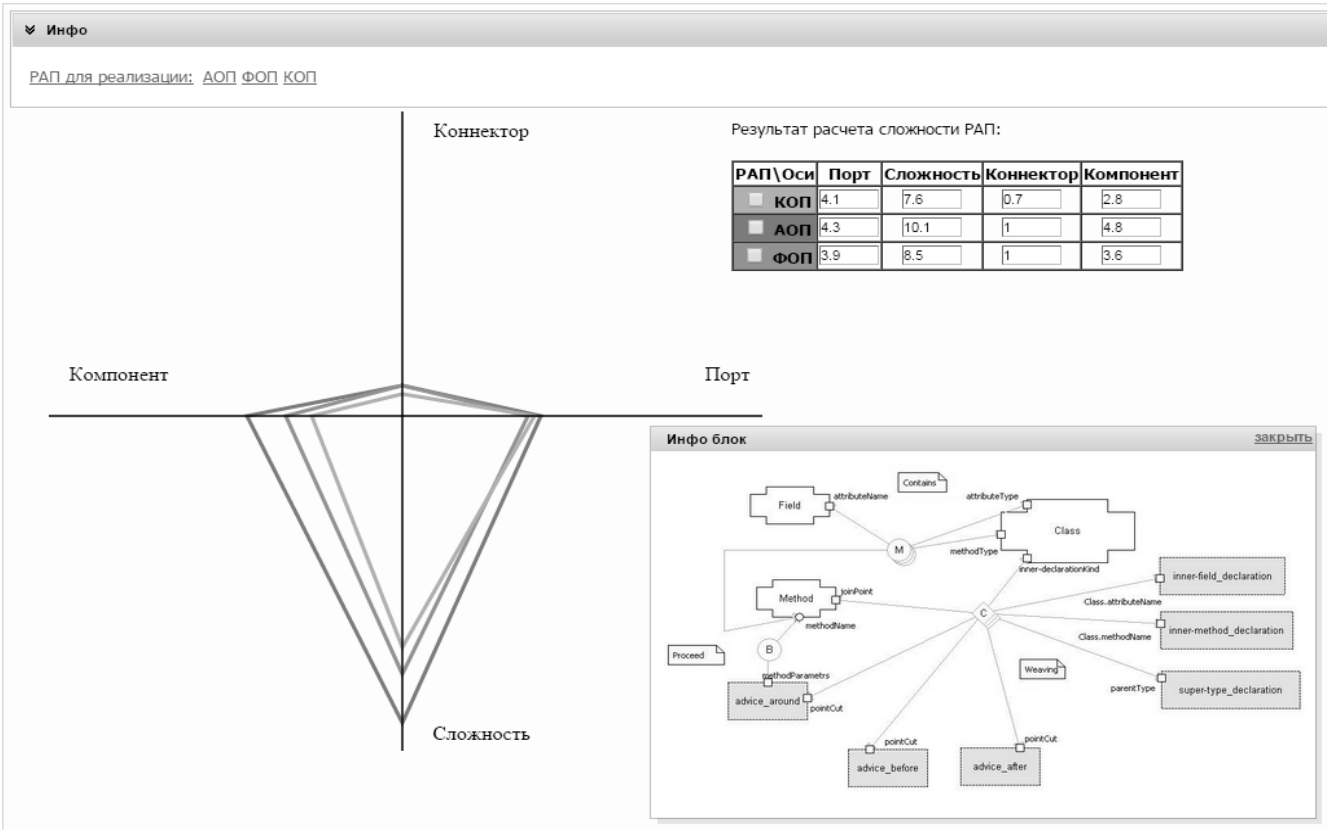


Рисунок 4.8 – Фрагмент інтерфейсу користувача «РАП»

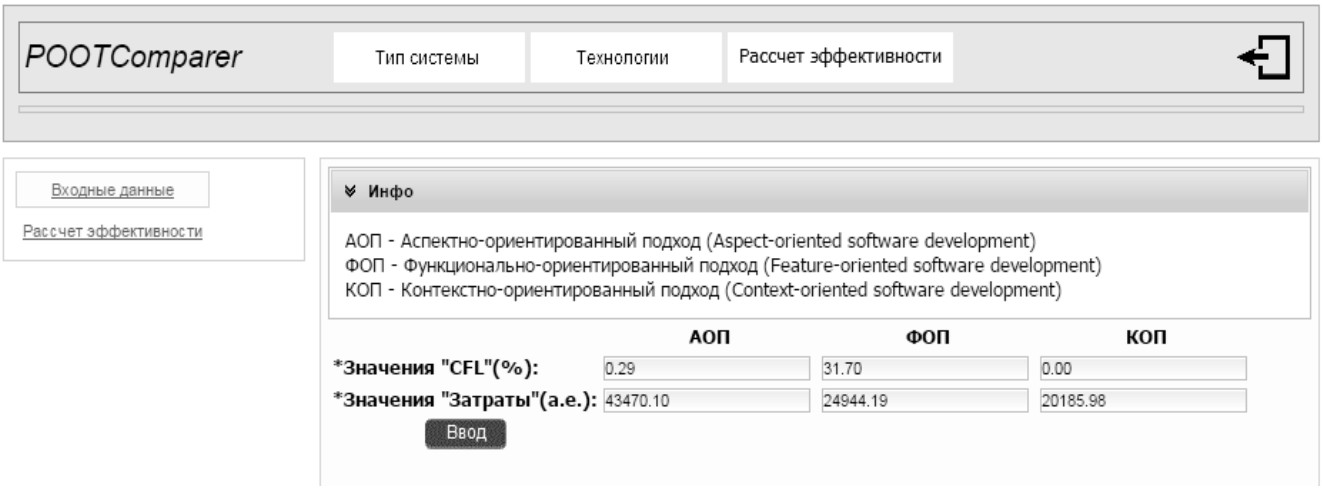


Рисунок 4.9 – Фрагмент інтерфейсу користувача
«Вхідні дані для розрахунку коефіцієнта ефективності»

Графічний компонент дає можливість змінювати кут огляду, та необхідним чином розвертати графік.

Рисунок 4.10 – Фрагмент інтерфейсу користувача
«Розрахунок коефіцієнту ефективності»

Таким чином, розроблений прототип CASE-засобу дозволяє автоматизувати усі основні фази розробленої інформаційної технології для оцінки ефективності використання ПООТ в процесі супроводу УПС (див. п. 3.5 попереднього розділу дисертації).

4.3. Методика проведення експерименту, вхідні дані та приклади розрахунків обчислювальних експериментів

Відповідно до розроблених у п.п. 3.1 – 3.4 загальної алгоритмічної моделі (див. у [150]) та окремих методів і процедур для оцінки ефективності використання ПООТ в процесі супроводу УПС, які об’єднуються у схемі запропонованої інформаційної технології (див. рис. 3.8), проведення обчислювального експерименту передбачає наявність вихідного об’єктно-орієнтованого програмного коду УПС та програмного коду 3-х модифікацій такої системи із використанням відповідних ПООТ [160]. Саме на підставі обробки цих програмних артефактів можуть бути отримані початкові експериментальні дані,

які потім мають бути структуровані у вигляді інформаційного базису алгоритмічної моделі процесу оцінки ефективності ПООТ (див. вирази (3.1) – (3.2)).

Слід зазначити, що для повного моделювання навіть одного інформаційного простору $П_2$ «Тип ПС» (див. вираз (3.4)), який містить 9 квадрантів для різних типів УПС, необхідно мати доступ до вихідного коду 9 таких УПС різної структурної складності та розробити додатково 27 ПООТ - модифікації їх вихідного коду, складність яких буде зростати пропорційно до початкової структурної складності кожної з таких систем. Ця обставина, а також необхідність урахування того факту, що здебільшого системи даного типу є комерційними програмними продуктами з закритим вихідним кодом, доступ до якого є досить ускладненим, накладають деякі обмеження на проведення відповідного експерименту.

Тому для проведення реального експерименту була використана редуційована інформаційна модель простору «Тип ПС», яка представлена на рис. 4.11.

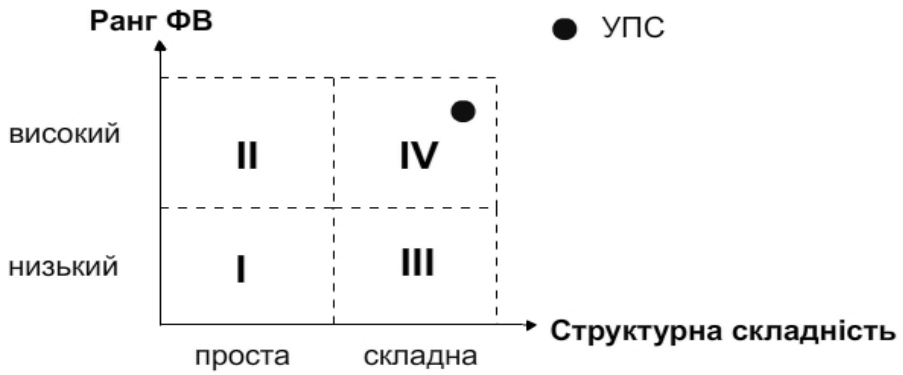


Рисунок 4.11 – Редуційований інформаційний простір «Тип ПС»

Вона містить 4 квадранти, які утворені віссю абсцис із двома значеннями ступеня структурної складності УПС: «проста», «складна»; та віссю ординат, із двома значеннями рангу вимог: «низький», «високий» (див. рис. 4.11).

Таким чином, зазначений підпростір для даного експерименту буде мати 4 типи УПС, де типи «I», «III» відповідають «стабільним УПС», ранг вимог яких низький; типи «II» та «IV» відповідають «не стабільним УПС», ранг вимог у яких високий, що вказує на постійні та складні модифікації таких систем. Точка на даному рисунку схематично відображає можливу позицію УПС в підпросторі $П_2$, тобто приналежність її до конкретного типу системи.

Відповідно до інформаційної технології, описаної у п. 3.5 попереднього розділу, обчислювальний експеримент складається з 6 основних етапів [147]:

1. Оцінка стану вимог до УПС.
2. Розрахунок структурної складності УПС.
3. Визначення типу системи.
4. Оцінка витрат на ПООТ-модифікацію вихідного коду УПС.
5. Оцінка ступеня присутності НФ в системі.
6. Остаточна оцінка ефективності використання ПООТ.

Для прикладу експериментальних розрахунків була обрана УПС, яка складається з 334 основних java-класів бізнес логіки, тобто класи в яких безпосередньо реалізована бізнес-логіка УПС, виключаючи службові (утилітарні класи) та класи, наявність яких зумовлена використанням того чи іншого каркасу розробки (framework), наприклад, таких як Struts Framework [170], Spring Framework [171] та відповідних каркасів об’єктно-реляційних відображень (object-relational mapping – ORM), для роботи з базами даних, такі як Castor [172] або Hibernate [173].

За період перших 6 місяців для оцінюваної УПС було зареєстровано 188 запитів на модифікацію. Пріоритет для вимог назначався експертним шляхом, відповідно до негайності мати реалізованою ту чи іншу вимогу, переважна більшість нових вимог мають пріоритет «негайний», або «актуальний». Оцінка складності вимог, згідно із запропонованою інформаційною технологією, проводилася із використанням відповідного засобу для автоматизації розрахунків

функціональної складності вимог, Function Point Modeler [169]. Після оцінок пріоритету та складності вимог, вони заносяться до відповідної СУВ: Rational Requisite Pro [109]. Для подальших розрахунків з використанням розробленого прототипу CASE-засобу, дані із цієї СУВ можуть бути експортовані у формат CSV (comma separated values).

Для розрахунку інтегрального показника стану вимог, а саме рангу вимог, використаний відповідний алгоритм [153], який запропонований у п. 3.2, попереднього розділу, згідно із ним, мають бути виконані наступні кроки:

Крок 1. Визначення початкових зон складності (ЗС) у просторі «Ранг вимог». Відповідно до даних отриманих після аналізу СУВ, а систем управляння задачами, доцільна кількість ЗС дорівнює 9 (відповідно до виразу (3.6)), тобто 3 рівня пріоритету виконання ФВ, та 3 рівня складності ФВ (див. рис. 3.1)

Крок 2. Для групування визначених ЗС у необхідну кількість кластерів, у даному випадку їх 2, що зумовлено редукцією підпростору $П_2$ «Тип ПС», застосована процедура кластеризації. На рис. 4.12 показане таке розбиття підпростору $П_3$. Відповідно до остаточного розбиття до кластеру К_I (зображений суцільними лініями), що відповідає за «прості» ЗС, відносяться такі зони: I {нейтральний пріоритет ФВ; низька складність ФВ}; II { нейтральний пріоритет ФВ; середня складність ФВ }; IV {актуальний пріоритет ФВ; низька складність ФВ}. Усі інші ЗС – є «складними» і відносяться до кластеру К_II (зображений пунктирними лініями).

Крок 3. Вагові коефіцієнти для кластерів, визначаються за методом MAI, та мають відповідні значення, для «простого» кластеру, $w_1 = 0.25$; для «складного» кластеру, $w_2 = 0.75$.

Крок 4. Кількість вимог у кластері К_I дорівнює 79, або 42,02%, кількість вимог, що відносяться до кластеру К_II, дорівнює109, або 57,98%. Відповідно до виразу (3.7), загальний ранг вимог дорівнює 101,5. У відсотковому співвідношенні розрахунок рангу вимогу представляється наступним чином:

$$RequirementRank = 42,02 \cdot 0,25 + 57,98 \cdot 0,75 = 10,51 + 43,49 = 54\%.$$

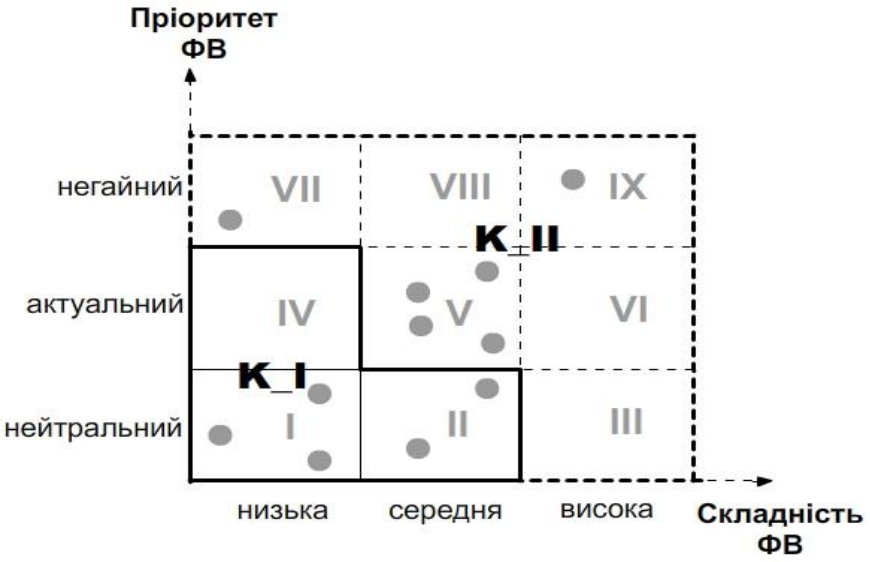


Рисунок 4.12 – Розбиття на кластери простору $П_3$ «Ранг вимог»

Крок 5. Межі шкали вимірювання для рангу вимог обчислюються відповідно до виразу (3.8), та за принципом, усі вимоги належать тільки одному кластеру.

$$RequirementRank \begin{cases} 100 \cdot 0,25 + 0 \cdot 0,75 = 25;\ Rank_d \in [0\%; 25\%) \\ 0 \cdot 0,25 + 100 \cdot 0,75 = 75;\ Rank_u \in [25\%; 75\%] \end{cases}.$$

Крок 6. Лінгвістична змінна приймає вигляд:

X – «Ранг ФВ»;

T(X) – {*простий, складний*},

U – [0;75], для побудови графіка, рис. 4.13 використовується кодована шкала [-6;6], відповідно до [122].

M – функція відповідності Харінгтона [122].

Таким чином, кількісний показник рангу вимог 54% (на рис. 4.13 відповідає показнику по осі абсцис 4,3 та d (4,3) = 0,98) відповідає терму «складний», тобто у редуційованому підпросторі $П_2$, дана УПС, буде мати значення Рангу ФВ «складний», що вказує на її не стабільність (див. рис. Б.4 у додатку Б).

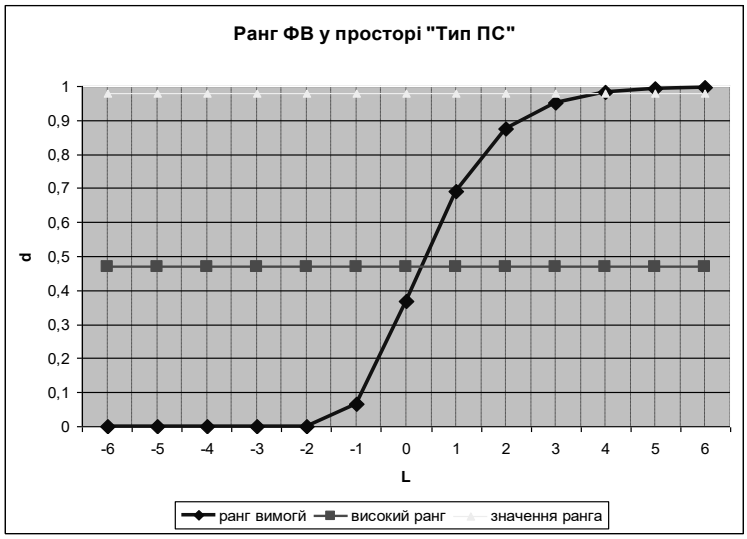


Рисунок 4.13 – Графік функції приналежності, лінгвістичної змінної «Ранг ФВ»

Другий етап експерименту передбачає розрахунок структурної складності УПС за відповідним алгоритмом, описаним у п.п. 3.2 попереднього розділу. УПС, що розглядається в контексті даного експерименту налічує 1013 java-класів, така кількість в тому числі зумовлена тим фактом, що система була розроблена на основі каркасу Struts із використанням ORM Castor. Зазначений алгоритм передбачає виконання 9-ти кроків.

Крок 1. Обчислення фактичних значень для кожної метрики об'єктно-орієнтованого коду, що представлені виразами ((2.6) – (2.13), (2.15))

Крок 2. Визначення коефіцієнтів, що є сталими для всіх систем: $w_1 = 0{,}04$; $w_2 = 0{,}01$; $w_3 = 0{,}06$; $w_4 = 0{,}11$; $w_5 = 0{,}13$; $w_6 = 0{,}22$; $w_7 = 0{,}22$; $w_8 = 0{,}21$.

Крок 3. Процентне співвідношення класів що потрапляють до зон складності, відповідно до граничних умов для кожної із 8-ми метрик коду, (див. наприклад [143, 156]): Так як простір типів систем $П_2$ редукований (див. рис. 4.11), то граничні умови, що відповідають цим зонам будуть об'єднані, тобто до зони А: «проста» відносяться всі елементи (класи, методи, пакети), показники яких не перевищують середнього граничного значення кожної із метрик (див. Табл. 4.3).

Таблиця 4.3.

Процентне співвідношення складності елементів

№	Позначення	Зона А (%)	Зона С (%)
1	V(G)	37	63
2	WMC	46	54
3	LOCOM	24	76
4	CBO	42	58
5	DIT	72	28
6	I	43	57
7	A	47	53
8	D	45	55

Крок 4. Вагові коєфіцієнти для зон А та С, відповідно будуть такими (за методом MAI): $p_1 = 0,4$; $p_3 = 0,6$.

Крок 5. Розрахунок показників складності, відповідно до виразу (3.9), для кожної метрики, див. Табл. 4.4.

Таблиця 4.4.

Складність елементів для метрик

№	Позначення	Зона А (%)	Зона С (%)	СМ
1	V(G)	14,8	37,8	52,6
2	WMC	18,4	32,4	50,8
3	LOCOM	9,6	45,6	55,2
4	CBO	16,8	34,8	51,6
5	DIT	28,8	16,8	45,6
6	I	17,2	34,2	51,4
7	A	18,8	31,8	50,6
8	D	18,0	33,0	51,0

Крок 6. Розрахунок максимальних показників для зон А, С відповідно до виразу (3.10): $Z_A = 0,4 \cdot 100 + 0,6 \cdot 0 = 40$; $Z_C = 0,4 \cdot 0 + 0,6 \cdot 100 = 60$.

Крок 7. Визначення граничних значень для показника структурної складності, відповідно до виразу (3.11). Верхній граничний показник для зони А: $SC_A = 40 \cdot (0,04+0.01+0.06+0.11+0.13+0.22+0.22+0.21)=40$. Верхній граничний

показник для зони С: $SC_C = 60 \cdot (0,04+0.01+0.06+0.11+0.13+0.22+0.22+0.21)=60$.

Тоді граничні значення для показника структурної складності будуть наступні:

$$StructuralComplexity \begin{cases} [0;40) \\ [40;60] \end{cases}$$

Крок 8. Визначення фактичного кількісного показника структурної складності.

$$SC = 52.6 \cdot 0,04+50.8 \cdot 0.01+55.2 \cdot 0.06+51.6 \cdot 0.11+45.6 \cdot 0.13+51.4 \cdot 0.22+50.6 \cdot 0.22+$$
$$+51.0 \cdot 0.21=2,104+0,508+3,336+5,676+5,928+11,308+11,132+10,71=50.702$$

Крок 9. Лінгвістична змінна приймає вигляд:

X – «Структурна Складність»;

T(X) – {*проста, складна*},

U – [0;60],

M – функція відповідності Харінгтона [122], її графічна інтерпретація подібна до графіку на рис. 4.8.

Таким чином, кількісний показник структурної складності $SC = 50.702$ відповідає терму «складна».

Третій етап експерименту передбачає типізацію УПС. Відповідно до результатів, отриманих на першому та другому етапах, можна однозначно визначити тип УПС, що досліджується. Вона має «складний» ранг ФВ та структурну складність «складну». Ця пара лінгвістичних значень відповідає типу системи «IV», див. рис. 4.11, це значить що система «не стабільна» та модифікація її структури досить витратна для команди супроводу (див. рис. Б.5 у додатку Б).

Четвертий етап експерименту передбачає обчислення ступеня присутності НФ, після ПООТ-модифікації вихідного коду УПС, для кожного із трьох підходів: АОП, КОП, ФОП, відповідно до виразу (2.20). В результаті аналізу вихідного коду даної системи було зареєстровано присутність наскрізної функціональності, при цьому (відповідно до виразу (2.20)) кількість уражених наскрізною функціональністю програмних класів основної бізнес–логіки C_{cf} = 158, кількість

класів вільна від присутності НФ складала 182 клас основної бізнес–логіки. Загальна кількість класів основної бізнес-логіки $N = 334$. На основі даних аналізу коефіцієнт питомої ваги НФ в УПС, що досліджується, $CFR = 46,71$. Цей показник відповідає ураженню середньої тяжкості, згідно до класифікації НФ, наведеної у п.п. 2.5, розділу 2. Для системи «IV» типу таке ураження є досить складним, зважаючи на постійну модифікацію, що викликана складним рангом вимог, та складною структурою вихідного коду.

Обчислення ступеня присутності НФ після ПООТ модифікацій дало наступні результати:

для АОП: $CFL = 1,12\%$;

для ФОП: $CFL = 31,70\%$;

для КОП: $CFL = 7,73\%$;

Результати обчислень показують, що в ПООТ-модифікаціях для підходів АОП та КОП наскрізна функціональність повністю ізольована в одному ПООТ-модулі. Результат застосування ФОП знижує ступінь присутності НФ в УПС, але наскрізна функціональність ізольована у декількох ФОП-модулях.

На п’ятому етапі експерименту була проведена оцінка витрат на ПООТ-модифікацію УПС, за принципом оцінки РАП. Тобто були виділені відповідні архітектурні конструкції, а саме: компоненти, конектори та порти, та визначена їх кількість, див. Табл. 4.5.

Таблиця 4.5.

Кількісні характеристики ПООТ модифікації

ПООТ	Компоненти (кількість)	Конектори (кількість)	Порти (кількість)	Витрати (а.е.)	Коефіцієнт РАП (а.е.)	Повні витрати (а.е.)
АОП	979	175	12965	4303,97	10,10	43470,10
ФОП	2816	3433	945	2934,61	8,50	24944,19
КОП	1736	954	3141	2656,05	7,60	20185,98

В ній три останніх стовпчика показують кількісні показники витрат, що обчислюються в архітектурних одиницях (а.е.). показники повних витрат на ПООТ-модифікацію (в останньому стовпчику), отримані відповідно до виразу (3.23).

На останньому етапі експерименту був обчислений показник ефективності використання ПООТ, який визначений виразом (2.5) (див п. 2.1 розділу 2) та розрахований за процедурою нечіткого виводу, що докладно описана у п.п. 3.4, попереднього розділу. Дані отримані за результатами процедури, показані у табл. 4.6.

Таблиця 4.6.

Коефіцієнт ефективності використання ПООТ

ПООТ	Витрати (а.е.)	Коефіцієнт присутності НФ $\boldsymbol{CFL}$ (%)	Коефіцієнт ефективності $\boldsymbol{E_{Koeff}}$ (%)
АОП	43470,10	1,12	36,30
ФОП	24944,19	31,70	29,60
КОП	20185,98	7,73	78,40

Як видно із Табл. 4.6, найбільший коефіцієнт ефективності (78,40%) з точки зору усунення негативних проявів НФ для даного типу УПС, забезпечує застосування технологія КОП. В той же час, вибір технології ФОП є найбільш нераціональним, з коефіцієнтом ефективності лише 29,60%. Таким чином, показано, що реалізація запропонованого підходу забезпечує можливість отримання кількісних оцінок щодо ефективності застосування окремих ПООТ для певного типу УПС, тобто дозволяє прийняти мотивоване рішення відносно вибору найбільш ефективної технології у процесі супроводу такої системи.

4.4. Аналіз отриманих результатів і практичні рекомендації по використанню ПООТ для супроводу УПС

На основі обробки повного об’єму експериментальних даних, приклад розрахунків яких був приведений у попередньому параграфі, були отримані остаточні результати застосування запропонованої інформаційної технології визначення ефективності використання ПООТ в процесі супроводу УПС [147, 160]. Для цього були обрані чотири УПС відповідного класу та сфери застосування. Початкові характеристики цих систем для першої фази експерименту наведені в Табл. 4.7.

Таким чином, УПС №1 відповідно до загальної процедури запропонованої інформаційної технології віднесена до першого (I) типу ПС, вона є стабільною з точки зору стану запитів на модифікацію, тобто стан її ФВ визначається як «низький», а ранг ФВ представляється найнижчим «нейтральним» пріоритетом, та «низькою» функціональною складністю (див. рис. 4.8).

Таблиця 4.7.

Вхідні характеристики досліджуваних УПС

№ УПС	Кількість java- класів що реалізують бізнес-логіку	Ранг ФВ	Структурна складність	Тип ПС
1	58	Низький	Проста	I
2	97	Високий	Проста	II
3	119	Низький	Складна	III
4	334	Високий	Складна	IV

Також УПС №1 має «просту» структурну складність, що вказує на те що показники по всім метрикам складності програмного коду не перевищують їх середнього граничного значення для більшості структурних елементів (методів, класів).

УПС №2 віднесена до другого (II) типу ПС, вона є «нестабільною» з точки зору стану запитів на модифікацію, тому що має «високий» ранг ФВ, що характеризується здебільшого максимальним «невідкладним» пріоритетом виконання вимог та максимальною «високою» функціональною складністю вимог. У той самий час, УПС №2 має «просту» структурну складність, що як у випадку із УПС №1, вказує на те, що показники метрик програмного коду не перевищують середні граничні умови для більшості структурних елементів цієї програмної системи.

УПС №3 віднесена до третього (III) типу ПС, вона є «стабільною» з точки зору стану запитів на модифікацію, т.я. має «низький» ранг ФВ, але вона має «високу» структурну складність, що вказує на те, що показники метрик програмного коду перевищують середні граничні умови, тобто належать до «складного» сегменту, для більшості структурних елементів цієї програмної системи. Це, в свою чергу, вказує на те, що УПС містить недосконалі проектні рішення та зумовлює ускладненість її супроводу.

УПС №4 віднесена до четвертого (IV), найскладнішого типу ПС, вона є «не стабільною» з точки зору стану запитів на модифікацію, тому що має «високий» ранг ФВ. УПС №4 має «високу» структурну складність, що вказує на те, що показники метрик програмного коду перевищують середні граничні умови, тобто належать до «складного» сегменту УПС з токи зору більшості структурних елементів цієї програмної системи. Відповідно, ця УПС містить недосконалі проектні рішення, які з урахуванням функціональної складності і невідкладності реалізації нових ФВ дуже сильно ускладнюють супровід такої УПС.

Для зазначених УПС, відповідно до четвертого етапу методики проведення експерименту (представленої у попередньому параграфі), був визначений коефіцієнт питимої ваги НФ (відповідно до виразу (2.18)), див. Табл. 4.8. Слід зазначити, що для розрахунку показників НФ, до розгляду приймаються незалежні від каркасу (framework) класи, а саме, java-класи бізнес–логіки.

Згідно із даними у Табл. 4.9 та відповідно до запропонованою класифікації НФ, яка наведена у п. 2.5, всі розглянуті вище УПС мають «ураження» НФ середньої тяжкості.

Таблиця 4.8.

Коефіцієнт питомої ваги в досліджуваних УПС

№ УПС	Загальна кількість класів бізнес-логіки	Кількість «уражених» класів бізнес-логіки	Кількість класів бізнес-логіки «вільних» від НФ	Показник коефіцієнта питомої ваги НФ
1	58	21	37	36,21%
2	97	59	51	60,82%
3	119	76	43	63,87%
4	334	156	178	46,71%

Для УПС III та IV, такі ураження є досить суттєвими, так як НФ має властивості «заплутуватись» та «перемішуватись» (див. п. 1.2 першого розділу роботи), що у поєднанні із складною структурою цих УПС, додатково ускладнюють процес супроводу системи. Після відповідних ПООТ - модифікацій вихідного коду усіх УПС, розрахований ступінь присутності НФ (*CFL* – Crosscutting Functionality Level) представлено графіками на рис. 4.14.

Із результатів розрахунку *CFL* видно, що найменший ступінь присутності НФ після ПООТ - модифікації структури УПС забезпечує застосування аспектно-орієнтованому підходу (АОП), який є близьким до 0,05%, тобто використання АОП дозволяє ізолювати НФ в найменшій кількості додаткових АОП - модулів.

Відповідно до п’ятого етапу методики проведення експерименту для зазначених УПС були розраховані витрати на їх відповідні ПООТ – модифікації (згідно виразу (3.22)), що вимірюються в архітектурних одиницях (a.e.), результати цих розрахунків зведені в Табл. 4.9.

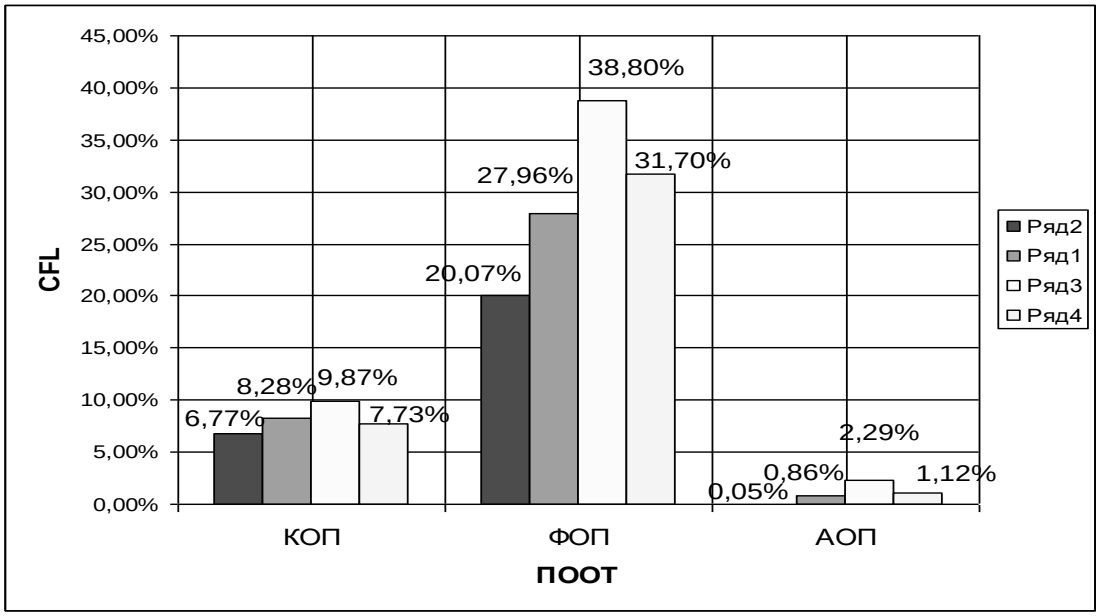


Рисунок 4.14 – Показник ступеню присутності НФ (CFL)

Дані в цій таблиці вказують на те, що найменш затратною за кількістю архітектурних одиниць є контекстно-орієнтований підхід (КОП), який для всіх типів УПС дає найменші показники архітектурної складності.

Таблиця 4.9.

Результат розрахунку затрат на ПООТ-модифікації УПС

ПООТ	Затрати на ПООТ-модифікацію, *Cost* (a.e.)			
	Тип Системы			
	I	II	III	IV
КОП	1933.90	3008.31	5956.42	20185.98
ФОП	3183.84	5264.56	7943.17	24944.19
АОП	5525,31	7941.73	17040.01	43470.10

Останній етап експерименту передбачає комплексну оцінку коефіцієнту ефективності (K_{Eff}, що вимірюється у відсотках «%») на основі попередньо розрахованих показників ступеня присутності НФ (CFL, що вимірюється у відсотках «%») та затрат на ПООТ-модифікацію (Cost, що вимірюються у архітектурних одиницях «a.e.»).

Результати цих розрахунків наведені у агрегованій Табл. 4.10, де у першій колонці відображаються досліджувані типи УПС, а інші три групи колонок групуються за трьома окремими ПООТ – підходами: КОП, ФОП та АОП.

Таблиця 4.10.

Результати розрахунку коефіцієнта ефективності ПООТ

Тип УПС	Затрати ПООТ-модифікації *Cost* (а.е.)			Ступінь присутності НФ *CFL* (%)			Коефіцієнт ефективності ПООТ K_{Eff} (%)		
	ПООТ			ПООТ			ПООТ		
	КОП	ФОП	АОП	КОП	ФОП	АОП	КОП	ФОП	АОП
I	1933.90	3183.84	5525,31	6,77	20,07	0,05	91,80	92,00	93,30
II	3008.31	5264.56	7941.73	8,28	27,96	0,86	92,30	91,30	92,90
III	5956.42	7943.17	17040.01	9,87	38,80	2,29	93,30	64,20	92,50
IV	20185.98	24944.19	43470.10	7,73	31,70	1,12	78,40	29,60	36,30

За результатами розрахунку коефіцієнту ефективності застосування ПООТ, можна зробити наступні висновки:

1. Для УПС, що належать до першого (I) та другого типів (II), немає суттєвої різниці між досліджуваними ПООТ - підходами, для всіх цих підходів $K_{Eff} > 90\%$;

   Причина: низька структурна складність таких УПС.

2. Для УПС третього (III) типу менш ефективний функціонально-орієнтований підхід (ФОП), який дає показник коефіцієнта ефективності 64,2 %.

Причина: значні відмінності у механізмах зв'язування ПООТ - модулів та класів базової бізнес-логіки.

3. Для УПС четвертого (IV) типу найбільш ефективним є контекстно-орієнтований підхід (КОП), коефіцієнт ефективності застосування якого дорівнює 78,4%. Цей факт має практичне значення для прийняття рішення щодо цільового вибору ПООТ для супроводу УПС.

Друга фаза експерименту передбачає дослідження кількості дефектів, що реєструються в УПС після відповідних ПООТ - модифікацій (див. п. 1.2 першого розділу), а саме, відповідно до припущення про наявність позитивної кореляції між рівнем присутності наскрізної функціональності та кількості дефектів в програмному коді УПС. Для цієї фази експерименту кількість дефектів у програмному коді УПС визначалася шляхом прямого вимірювання в процесі її супроводу за наступною формулою:

$$NoD = Count(deffects) \tag{4.1}$$

де *NoD* (Number of Defects) – кількість дефектів;

Count (defects) – відповідна функція визначення кількості дефектів.

Таблиця 4.11.

Кількість дефектів в програмному коду УПС

Тип УПС	Початкова кількість дефектів в УПС (до застосування ПООТ) $NoD^{(ООП)}$			Кількість дефектів в програмному коду після застосування ПООТ $NoD^{(ПООТ)}$			Залишковий рівень дефектів (%)		
	ПООТ			ПООТ			ПООТ		
	КОП	ФОП	АОП	КОП	ФОП	АОП	КОП	ФОП	АОП
I	28	28	28	6	4	4	21,4	14,3	14,3
II	68	68	68	19	24	14	27,9	35,3	20,6
III	46	46	46	13	22	16	28,8	47,8	34,7
IV	83	83	83	18	56	27	21,7	67,4	32,5

У Табл. 4.11 приведені результати прямого вимірювання дефектів за 6 місяців 2015 року, під час тестування УПС без змін у програмному ООП-коді та їх відповідних ПООТ-модифікацій. Дефекти були зареєстровані у системі управління дефектами (Bug Tracking System) для різних типів досліджуваних УПС. Зниження кількості дефектів більше ніж на 30%, свідчить про наявність

позитивної кореляції між кількістю дефектів та відповідними значеннями коефіцієнту ефективність застосування ПООТ в процесі супроводу УПС, отриманими на першій фазі цього експерименту.

Висновки по розділу 4

У цьому розділі роботи отримані наступні результати:

1. Надано детальний опис предметної області (ПрО) застосування конкретного класу УПС, для яких в подальшому проводиться дослідження працездатності та ефективності запропонованого підходу. При цьому визначені топологія організаційної побудови основних об’єктів та інформаційних процесів у цій ПрО, наведена архітектура програмних компонентів і представлено детальний опис технічних характеристик для цього класу УПС.

2. Розроблені основні варіанти сценаріїв використання, компонентна архітектура, схема бази даних та основні візуальні інтерфейси користувача інструментального CASE-засобу для реалізації запропонованої інформаційної технології.

3. Представлена методика проведення обчислювальних експериментів, вхідні дані та приклади виконання розрахунків окремих етапів.

4. Наведені результати аналізу проведених експериментальних досліджень та сформульовані практичні рекомендації щодо ефективного застосування існуючих ПООТ для супроводу окремих типів УПС.

Нові наукові результати, викладені в цьому розділі, опубліковані в роботах [147, 150, 160, 166].

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дисертаційній роботі поставлена та вирішена актуальна науково-практична задача розробки моделей та інструментальних засобів для підвищення ефективності використання пост об’єктно-орієнтованих технологій (ПООТ) у процесі супроводу успадкованих програмних систем (УПС). Внаслідок виконаних досліджень отримано наступні наукові та практичні результати:

1. Проаналізовано особливості процесу супроводу та структурної модифікації УПС, які розроблені на основі ООП, із урахуванням проблем, що виникають при появі ефекту наскрізної функціональності (НФ), а також проведено порівняльній аналіз основних існуючих ПООТ, а саме: аспектно-орієнтованого підходу (АОП), функціонально-орієнтованого підходу (ФОП) та контекстно-орієнтованого підходу (КОП), з метою вивчення особливостей структурних механізмів ізоляції та усунення негативних проявів НФ у УПС.

2. Формалізовано поняття ефективності застосування ПООТ в процесі супроводу УПС і запропоновано знання-орієнтований методологічний підхід для комплексної оцінки ефективності застосування ПООТ в процесах супроводу ПС.

3. Вперше розроблена алгоритмічна модель процесу визначення ефективності застосування ПООТ при супроводі УПС, яка відрізняється від існуючих комплексним застосуванням багатовимірного інформаційного ресурсу, експертних методів та відповідних метрик, що дозволяє кількісно оцінити ступінь ефективності застосування окремих ПООТ у залежності від суттєвих характеристик УПС.

4. Отримали подальший розвиток методи та інструментальні засоби аналізу та визначення стану УПС у процесі її супроводу шляхом використання сукупності обчислювальних алгоритмів і кількісних критеріїв, що дозволяє враховувати як структурну складність так і динаміку змін у вимогах до певної УПС в процесі її супроводу.

5. Набули подальшого розвитку моделі та інструментальні засоби дослідження технологічних особливостей використання ПООТ в процесах супроводу УПС шляхом розробки еталонних архітектурних моделей окремих ПООТ, що дає можливість кількісно визначити питомі витрати при їх застосуванні для програмної модифікації УПС з метою усунення негативних наслідків присутності НФ.

6. Удосконалено інформаційну технологію оцінки ефективності застосування ПООТ в процесах супроводу УПС за рахунок використання методів нечіткого виводу у поєднанні із кількісними метриками оцінки як загального рівня присутності, так і ступеня розповсюдженості НФ в програмному коді УПС, що забезпечує можливість попереднього аналізу та оцінки ефективності альтернативних варіантів проведення модернізації УПС.

7. Із використанням вільно поширюваних програмних засобів і технологій таких як Java, JSP, Spring MVC, MySQL, Castor JDO та XML реалізовані основні програмні компоненти інструментального CASE-засобу для реалізації запропонованого підходу.

8. Проведено експериментальне дослідження запропонованого підходу та на основі аналізу отриманих результатів розроблені практичні рекомендації щодо підвищення ефективності використання ПООТ в процесах супроводу УПС, ефективність використання ПООТ для зменшення рівня присутності НФ у програмному коді становить більш ніж 29,6%. Також встановлено, що в кінцевому рахунку застосування ПООТ не тільки зменшує рівень присутності НФ в програмних модулях бізнес-логіки УПС, але й зменшує кількість дефектів більше ніж на 30%.

9. Одержані в роботі теоретичні та практичні результати, були використані при виконанні держбюджетної теми у НТУ «Харківський політехнічний інститут», а також при виконанні ІТ-проектів в компаніях Bit media e-learning solution GmbH & Co KG (м. Грац, Австрія) та ТОВ «ІНТЕРПАК ІНФОРМАЦІЙНІ СИСТЕМИ» (м. Харків).

10. Отримані в дисертаційній роботі результати також були запроваджені в навчальному процесі кафедри програмної інженерії та інформаційних технологій управління НТУ «ХПІ» в дисциплінах «Основи проектування ПЗ», «Аналіз вимог до ПЗ», «Методи та засоби автоматизації процесів життєвого циклу ПЗ».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Systems and software engineering – Software Life Cycle Process: ISO/IEC 12207:2008. – 2008. – 123 p.

2. Software Engineering — Software Life Cycle Processes — Maintenance: ISO/IEC14764-2006. – 2006. – 44 p.

3. Erdil, K., et al. Software Maintenance As Part of the Software Life Cycle / Erdil K., Finn E., Keating K., Meattle J., Park S., Yoon D. – Department of Computer Science, Tufts University, Medford, Massachusetts, USA, 2003. – 49 p.

4. Соммервил И. Инженерия программного обеспечения. 6–е издание / Соммервил И.; пер. с англ. – М.: Вильямс, 2002. – 624 с.

5. Kim, H. S., Bieman J. Migrating legacy systems to CORBA based distributed environments through an automatic wrapper generation technique. / Kim, H. S., Bieman J.// Proceedings of the SCI’2000: the 4th World Multiconference on Systemics Cybernetics and Informatics and the ISAS’2000: the 6th International Conference on Information Systems Analysis and Synthesis”.

6. Kuipers, T. Techniques for Understanding Legacy Software Systems. Aacademisch proefschrift, ter verkrijging van de graad van doctor / Kuipers T. – University of Amsterdam, Amsterdam, Nederland, February 2002. – 154 p.

7. Stroustrup, B. The C++ programming language. Fourth edition / Stroustrup B. – Michigan, U.S.: Addison-Wisley, 2013. – 1346 p.

8. C# Language Specification [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/en-us/library/ms228593.aspx.

9. Java Software. Офіційний сайт розробника [Електронний ресурс] – Режим доступу: https://www.oracle.com/java/index.html.

10. Нойбург, М. Программирование для iOS 7. Основы Objective-C, Xcode и Cocoa / Нойбург М. – М.: Вильямс, 2014. – 384 с.

11. The D Programming Language. Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://dlang.org.

12. X++ Language Programming Guide [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/en-us/library/aa867122.aspx.

13. Swift. Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: https://developer.apple.com/swift/.

14. Microsoft Windows OS, Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: https://www.microsoft.com/uk-ua/windows.

15. Linux OS. Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: http://www.linux.org.

16. Apple OS X. Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: http://www.apple.com/osx/.

17. Apple iOS. Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: http://www.apple.com/ios/.

18. Android OS. Офіційний сайт розробника розробника [Електронний ресурс] – Режим доступу: http://www.android.com/.

19. Gamma, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software / Gamma E., Helm R., Johnson R., Vlissides J. – Addison-Wesley, 1994. – 395 p.

20. Eder, J., Kappel, G., Schrefl, M. Coupling and Cohesion in Object-Oriented Systems. Technical Report. / Eder J., Kappel G., Schrefl M. – University of Klagenfurt, 1994. – 34p.

21. Sheldon, T., Jerath, Kh., Chung, H. Metrics for Maintainability of Class Inheritance Hierarchies. / Sheldon T., Jerath Kh., Chung H. // Journal of Software Maintenance and Evolution. – 2002. – vol. 14. – pp. 1-14.

22. Harrison, R., Counsell, S.J. The Role of Inheritance in the Maintainability of Object-Oriented Systems. / Harrison R., Counsell S.J. // Proceedings of the ESCOM ’98: European Software Control and Metrics Conference. – 1998. – pp. 449-457.

23. Taylor, W.E., Haddad, H.M. Flaws of the OO paradigm: a perspective from the procedural side. / Taylor W.E., Haddad H.M. // Journal of Computing Sciences in Colleges. – 2012. – vol. 28(2). – pp. 160-167.

24. Brown, W. J., Malveau, R. C., et al. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. / Brown W. J., Malveau R. C., McCormick H. W., Mowbray T. J. – USA: John Wiley & Sons, Inc, 1998. – 336 p.

25. Ткачук, Н.В., Нагорный, К.А., Структурная адаптация программных систем: анализ состояния проблемы и некоторые подходы к ее решению / Ткачук Н.В., Нагорный К.А. // Східно-Европейський журнал передових технологій. – 2009. – № 6 (42). – с. 33-36.

26. Herring, C., Kaplan, S. The Viable System Model for Software. / Herring C., Kaplan S. // Proceedings of the SCI'2000: the 4th World Multi-conference on Systems Cybernetics and Informatics. - Orlando, Florida. – 2000.

27. Miller, S.D., DeCario, R.A., Mathur, A.P. A Software Cybernetic Approach to Control of Software Systems / Miller S.D., DeCario R.A., Mathur A.P. // Proceedings of the 29th Annual International Computer Software and Applications Conference. – 2005.

28. Расстригин, Л.А. Адаптация сложных систем. Методы и приложения / Расстригин Л.А. – Рига: Зинатне, 1981. – 375 с.

29. Tempero, E.; Yang, H. Y.; Noble, J. What programmers do with inheritance in Java. / Tempero E.; Yang H. Y.; Noble J. // Proceedings of the ECOOP 2013: the 27th European Conference on Object-Oriented Programming. – 2013. – pp. 577–601.

30. Bloch, J. Effective Java. 2nd edition / Bloch J. – Stoughton, Massachusetts, US: Addison-Wesley, 2008. – 346 p.

31. Nystrom, N., Chong, S., Myers, A. Scalable extensibility via nested inheritance. / Nystrom, N., Chong, S., Myers, A. // Proceedings of the OOPSLA ’04: the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. – 2004.

32. Frost, C. Millstein, T. Modularly typesafe interface dispatch in JPred. / Frost C. Millstein T. // Proceedings of the FOOL/WOOD’06: An International Workshop on Foundations and Developments of Object-Oriented Languages. – 2006/

33. Martin, R. C. Java and C++: a critical comparison. In Java Gems: jewels from Java Report / Martin, R. C. – Cambridge University Press, February1998. – pp. 51–68.

34. Kaur, A., Johari, K. Identification of Crosscutting Concerns: A Survey. / Kaur A., Johari K. // International Journal of Engineering Science and Technology. – 2009. – vol. 1(3). – pp. 166-172.

35. Mantis bug tracker [Електронний ресурс] – Режим доступу: https://www.mantisbt.org, 08.2015.

36. Revelle, M. Broadbent, T., Coppit, D. Understanding Concerns in Software: Insights Gained from Two Case Studies. / Revelle M. Broadbent T., Coppit D. //, Proceedings of the International workshop on program comprehensio. – 2005.

37. Rosenhainer. L. Identifying Crosscutting Concerns in Requirements Specifications. / Rosenhainer. L. // Proceedings of the OOPSLA’04 “Early Aspects 2004”: AspectOriented Requirements Engineering and Architecture Design Workshop. – Vancouver, Canada. – 2004.

38. Kandé, M. M., Strohmeier, A. Modeling Crosscutting Concerns using Software Connectors. / Kandé M. M., Strohmeier A. // Proceedings of the ASoC’01. Tampa Bay, Florida. – 2001.

39. Marin, M., Moonen, L., van Deursen, A. A Classification of Crosscutting Concerns. / Marin M., Moonen L., van Deursen A. // Proceedings of the ICSM 2005: the IEEE International Conference on Software Maintenance. – 2005. – pp. 673–676.

40. Figueiredo, E., Silva, B., et al. Crosscutting Patterns and Design Stability: An Exploratory Analysis / Figueiredo E., Silva B., Sant’Anna C., Garcia A., Whittle J., Nunes D. // Proceedings of the ICPC 2009: th 17th International Conference on Program Comprehension. – 17-19 May 2009.

41. Aversano, L., Cerulo, L., Penta, M. Di. Relating the Evolution of Design Patterns and Crosscutting Concerns. / Aversano L., Cerulo L., Penta M. Di. // Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation. – 2007. – pp. 180-192.

42. Офіційний веб-сайт MSDN [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/en-us/library/ee658105.aspx, 08.2015.

43. Clarket, S., et al. Separating Concerns throughout the Development Lifecycle. / Clarket S., et al. // Proceedings of the ECOOP 1999: International Workshop on Aspect-Oriented Programming. – 1999.

44. Krinke, J. Mining execution relations for crosscutting concerns. / Krinke. J. // IET Software. – April 2008. – vol. 2(2). – pp. 65–78.

45. Apel, S., Batory, D., Rosenmüller, M. On the Structure of Crosscutting Concerns: Using Aspects or Collaborations? / Apel S., Batory D., Rosenmüller M. // Proceedings of the AOPLE'06: the 1st Workshop on Aspect-Oriented Product Line Engineering co-located with the GPCE'06: the 5th International Conference on Generative Programming and Component Engineering. – Portland, OR, USA. – Oct. 2006.

46. Ткачук, Н.В., Нагорный, К.А. Об одном подходе к оценке эффективности применения пост объектно-ориентированных технологий при сопровождении программных систем / Ткачук Н.В., Нагорный К.А. // Проблемы программирования. – К.: НАН України. – 2010. – № 2-3 (спец. выпуск). – с. 252 - 260.

47. Taromirad, M., Paige, M. Agile Requirements Traceability Using Domain-Specific Modeling Languages. / Taromirad M., Paige M. // Proceedings of the Extreme Modeling Workshop. – 2012. – pp. 45-50.

48. Aversano, L., Cerulo, L., Penta, M. Di. The Relationship between Design Patterns Defects and Crosscutting Concern Scattering Degree: An Empirical Study. /

Aversano L., Cerulo L., Penta M. Di. // Journal IET Software. – 2009. – vol. 3. – pp. 395 - 409.

49. IEEE Standard Classification for Software Anomalies: IEEE 1044-2009. – 2010. – 15 p.

50. Eaddy, M. et al. Do Crosscutting Concerns Cause Defects? / Eaddy, M. et al. // Journal IEEE Transactions Software Engineering/ – 2008. – vol. 34(4). – pp. 497-515.

51. Tarr, P.L., et al. N Degrees of Separation: Multi-Dimensional Separation of Concerns. / Tarr P.L., et al. // Proceedings of the ICSE 1999: the International Conference on Software Engineering. Los Angeles, USA. – 1999. – pp. 107-119.

52. Ткачук Н.В. Архитектуры, модели и технологии программного обеспечения информационно-управляющих систем: Монография: сборник научных трудов / Н.В. Ткачук, В.А. Шеховцов, Д.В. Кукленко, В.Е. Сокол; [Под ред.: М.Д. Годлевский]; М-во образования и науки Украины. – Х.: НТУ «ХПИ». – 2005. – 546 с.

53. Przybyiek, A. Post Object-oriented Paradigms in Software Development: A Comparative Analysis. / Przybyiek, A. // Proceedings of the International Multi-conference on Computer Science and Information Technology. – 2007. – pp. 1009-1020.

54. Вэб-сайт офіційного співтовариства АОП (Official Web-site Aspect-oriented Software Development community) [Електронний ресурс] – Режим доступу: http://aosd.net

55. Вэб-сайт офіційного співтовариства ФОП (Official Web-site of Feature-oriented Software Development community) [Електронний ресурс] – Режим доступу: http://fosd.de

56. Вэб-сайт групи розробки КОП (Web-site of Context-oriented Software Development group), http://www.hpi.uni-potsdam.de/hirschfeld/cop, 08.2015.

57. Конференція з питань АОП «Modularity 2015», http://www.aosd.net/2015/, 08.2015.

58. Конференція з питань ФОП «FOSD Meeting 2015» [Електронний ресурс] – Режим доступу: http://www.fosd.net/2015.

59. Семінар з питань КОП «7th International Workshop on Context-Oriented Programming» [Електронний ресурс] – Режим доступу: http://2015.ecoop.org/track/COP-2015-papers

60. Європейська конференція з об'єктно-оріентованого програмування «European Conference on Object-Oriented Programming – ECOOP 2015» [Електронний ресурс] – Режим доступу: http://2015.ecoop.org

61. Xerox PARC, офіційний сайт [Електронний ресурс] – Режим доступу: http://www.parc.com

62. AspectJ, Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://www.eclipse.org/aspectj

63. AspectC++, Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://www.aspectc.org

64. PostSharp, Офіційний сайт розробника [Електронний ресурс] – Режим доступу: https://www.postsharp.net/aspects

65. AspectJS, Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://www.aspectjs.com.

66. Kiczales, G., et al. Aspect-Oriented Programming. / Kiczales G., et al. // Proceedings of the European Conference on Object-Oriented Programming. Springer-Verlag., Finland. – 1997.

67. Kiczales, G., et al., Aspect-Oriented Programming. Final Technical Report / Kiczales G., Hugunin J., Hilsdale E., Kersten M., Palm J., Lopes C., Griswold B., Isberg W. – Palo Alto Research Center, California, 2003. – 149 p.

68. Ткачук М.В. Кібернетичний підхід до адаптації програмних компонентів з використанням аспектно–орієнтованої технології / М. В. Ткачук, К. А. Нагорний, І. О. Мартінкус // Матеріали XVII міжн. науково-практ. конференції

«Інформаційні технології: наука, техніка, технологія, освіта, здоров'я», НТУ «ХПІ», Харків, 20-22 травня 2009 р. – с. 34.

69. Laddad, R. AspectJ in Action. Second Edition. Enterprise AOP with Spring Applications / Laddad, R. – US: Manning Publications Co., 2010. – 568 p.

70. Apel, S. The Role of Features and Aspects in Software Development. Dissertation, zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.) / Apel S. – Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany, March 2007. – 147 p.

71. FeatureIDE. Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide

72. Офіційний веб-сторінка засобу розробки ATS [Електронний ресурс] – Режим доступу: http://www.cs.utexas.edu/users/schwartz/ATS.html

73. FeatureC++, Офіційний сайт розробника [Електронний ресурс] – Режим доступу: http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec

74. ,FeatureHouse. Офіційний сайт розробника. [Електронний ресурс] – Режим доступу: http://www.fosd.de/fh

75. Batory, D. Feature-oriented programming and the AHEAD tool suite. / Batory D. // Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society. Washington, DC, USA. – 2004. – pp. 702–703.

76. Ткачук М. В. Розробка адаптивних програмних систем на основі функціонально-орієнтованого підходу. / М. В. Ткачук, К. А. Нагорний, М. М. Литвинчук // Матеріали XVII міжнародної науково-практичної конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров'я», НТУ «ХПІ», Харків, 20-22 травня 2009 р. – с. 33.

77. Hirschfeld, R., Costanza, P., Nierstrasz, O. Context-oriented Programming. / Hirschfeld R., Costanza P., Nierstrasz O. // Journal of Object Technology. – March–April 2008. – vol. 7, no. 3. – pp.125–151.

78. Context-oriented programming. Сайт розробника [Електронний ресурс] – Режим доступу: http://www.swa.hpi.uni-potsdam.de/cop/implementations/index.html

79. Costanza, P., Hirschfeld, R. Language Constructs for Context-oriented Programming An Overview of ContextL. / Costanza, P., Hirschfeld, R. // Proceedings of the 2005 Dynamic Languages Symposium. ACM Press. – October 2005.

80. Ткачук М. В., Нагорний К. А. Про один підхід до структурної адаптації програмних систем на основі пост об'єктно-орієнтованих технологій. Материалы XI международной научно-технической конференции «Системный анализ и информационные технологии», НТУУ «КПИ», Киев, 26-30 мая 2009 г. – с. 577.

81. Палагин А. В. Знание-ориентированные системы с обработкой естественно-языковых объектов: основы методологии и архитектурно-структурные организация / А. В. Палагин, С. Л. Кривой, Н. Г. Петренко // УСиМ - 2009, №3. – С. 42-55.

82. Палагин А. В. К вопросу системно-онтологической интеграции знаний предметной области / А. В. Палагин, Н. Г. Петренко // Математичні машини і системи. – 2007, №3-4. – С. 63-75.

83. Official Web-site of System Thinking World community [Електронний ресурс] – Режим доступу: http://www.systems-thinking.org/kmgmt/kmgmt.htm

84. Гамзаєв Р. О. Моделі та інформаційна технологія трасування вимог в гнучких процесах розробки програмного забезпечення [Текст]:. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 – інформаційні технології / Гамзаєв Рустам Олександрович. – Х.: НТУ ХПІ, 2013. – 173 с.

85. Сокол В. Є. Моделі та інструментальні засоби систем управління ІТ-інфраструктурою організацій [Текст]: дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 – інформаційні технології / Владислав Євгенович Сокол. – Х.: ХНУРЕ, 2014. – 179 с.

86. Tkachuk, M., Nagorniy, K., Gamzayev, R. Knowledge-based Approach to Effectiveness Estimation of Post Object-oriented Technologies in Software

Maintenance. / Tkachuk M., Nagorniy K., Gamzayev R. // Proceedings of the ICTERI-2015: the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, Lviv, May 14-16, 2015. – Vol-1356. – pp. 62-77.

87. Поморова, О. В., Говорущенко, Т. О. Аналіз методів та засобів оцінки якості програмних систем / О. В. Поморова, Т. О. Говорущенко // Радіоелектронні і комп'ютерні системи. – 2009. – № 6 (40). – с. 148-158.

88. Говорущенко Т.О. Дослідження відомих моделей оцінювання характеристик програмного забезпечення / Т. О. Говорущенко // Вісник Хмельницького національного університету – Хмельниціький: ХНУ. – 2013. – №1, с. 117 – 121.

89. Kearney, J. K. Software complexity measurement. / Kearney, J. K. //, Communications of the ACM. – 1986. – vol. 29. – pp.1044-1050.

90. Roca, J.L. A new entropy based method for computing software structural complexity. Technical report ARN-PI-2002-8 / Roca, J.L. – Autoradad Regulatoria Nuclear, Buenos Aires, Argentina, 2002. – 42 p.

91. Sadoui, L. Badri, M., Badri, L. Improving Class Cohesion Measurement: Towards a Novel Approach Using Hierarchical Clustering. / Sadoui L. Badri M., Badri L. // Journal of Software Engineering and Application. – 2012. – vol 5. pp. 449-458.

92. Коберн А. Быстрая разработка программного обеспечения / Коберн А. – М.: Лори, 2002. – 336 с.

93. Software and systems engineering - Software measurement - IFPUG functional size measurement method: ISO/IEC 20926:2009. – 2009. – 24 p.

94. Software engineering – COSMIC: A functional size measurement method: ISO/IEC 19761:2011. – 2011. – 14 p.

95. Software Engineering – Nesma Functional Size Measurement Method Version 2.1 – Definitions And Counting Guidelines For The Application Of Function Point Analysis: ISO/IEC 24570:2005. – 2005. – 124 p.

96. Boehm, B., Abts, C., et al. Software Cost Estimation with COCOMO II / Boehm B., Abts C., Brown A., Chulani S., Clark B., Horowitz E., Madachy R., Reifer D., Steece B. – New Jersey, USA: Prentice-Hall, 2000. – 540 p.

97. Banker, R.D., Kaufman, R.J., Kumar, R. An empirical test of object-oriented output measurement metrics in a computer aided software engineering (CASE) environment. / Banker R.D., Kaufman R.J., Kumar R. // Journal of Management Information Systems. – Winter 1991-1992. – vol. 8, no. 3. – pp. 127-150.

98. Stensrud, E. Estimating with enhanced object points vs. function points. / Stensrud E., editors: Boehm B., Madachy R. // Proceedings of the COCOMO'13: the 28th International Forum on COCOMO and Systems/Software Cost Modeling. USC Center for Software Engineering, Los Angeles CA. – 1998.

99. Aggarwal, K. K. Empirical Study of Object-Oriented Metrics. / Aggarwal K. K. // Journal Of Object Technology. – 2006. – vol. 5, no. 8. – pp. 149–173.

100. Aggarwal, K. K. Software Design Metrics for Object-Oriented Software / Aggarwal K. K. // Journal Of Object Technology. – 2006. – vol. 6, no. 1. – p. 121–138.

101. Nguyen, V., Deeds-Rubin, S., Tan, T., Boehm, B. A SLOC counting standard. Technical report / Nguyen V., Deeds-Rubin S., Tan T., Boehm B. – University of Southern California: Center for Systems and Software Engineering, California, USA, 2007. – 16 p.

102. Gronback, R. C. Software Remodeling: Improving Design and Implementation Quality. Using audits, metrics and refactoring in Borland® Together ® ControlCenter™. A Borland White Paper / Gronback, R. C. – January 2003. – 40 p.

103. Robert, C. M. Design Principles and Design Patterns. / Robert C. M. // www.objectmentor.com. – 2000. – p.23.

104. Systems and software engineering – Vocabulary: ISO/IEC 24765:2010. – 2010. – 410 p.

105. Вигерс К. Разработка требований к программному обеспечению / Вигерс К.; пер. с англ. – М.: Русская Редакция. – 2004. – 576 с.

106. International Institute of Business Analysis, A guide to the Business Analysis Body Of Knowledge (BABOK Guide). Version 2.0. – Toronto, Ontario, Canada, 2009. – 265 p.

107.Guide to the Software Engineering Body of Knowledge. Version 3.0, 2014 edition / Bourque P., Fairley R. E. (eds.) – IEEE Computer Society, Los Alamitos, CA, USA, 2014. – 335 p.

108. 830-1998 – IEEE Recommended Practice for Software Requirements Specifications. – 1998. – 40p

109. Система управляння вимогами, IBM Rational RequisitePro [Електронний ресурс] – Режим доступу: https://www-01.ibm.com/marketing/iwm/tnd/search.jsp?pn=Rational+RequisitePro

110. Система управляння вимогами, Accompa: Requirements Management Software Tool [Електронний ресурс] – Режим доступу: http://web.accompa.com/requirements-management-software

111. Система управляння вимогами, IBM DOORS Next Generation [Електронний ресурс] – Режим доступу: http://www-03.ibm.com/software/products/en/ratidoorng

112. Система управління вимогами, Gatherspace: Requirements Management Software, http://www.gatherspace.com.

113. IBM, Get it right the first time: writing better requirements. – 2009. – 60 p.

114. Clemmons, R. Project estimation with use case points. CrossTalk. / Clemmons, R. //The Journal of Defense Software Engineering. – February 2006. – pp. 18 - 22.

115. Henderson-Sellers, B., Zowghi, D., et al. Sizing Use Cases : How to create a standard metrical approach. / Henderson-Sellers, B., Zowghi, D., et al. // Object Oriented Information Systems. – 2002. – pp 409–421.

116. Information technology – software measurement – functional size measurement. Part 1 : definition of concepts : ISO/IEC 14143.1:2007. – 2007. – 6 p.

117. Santillo, L., Meli, R. Early Function Points: some practical experiences of use. / Santillo L., Meli R. // Proceedings of the ESCOM – ENCRESS 98: Project Control for 2000 and Beyond. – Rome, Italy. – May 27-29, 1998. – 13 p.

118. Santillo, L. Early FP Estimation and the Analytic Hierarchy Process. / Santillo L. // Proceedings of the ESCOM-SCOPE 2000. – Munich, Germany. – April 2000, – pp 249–257.

119. Hsu, C., Sandford, B.A. The Delphi Technique: Making Sense of Consensus. / Hsu C., Sandford B.A. // Practical Assessment, Research & Evaluation – 2007. – vol. 12(10). – pp 1-8.

120. IDEF. Integrated definition methods [Електронний ресурс] – Режим доступу: //http://www.idef.com

121. Заде, Л. Нечеткая логика: Понятие лингвистической переменной и его применение к принятию приближенных решений / Заде Л. – М.: Мир, 1976. – 167с.

122. Адлер Ю. П. Планирование эксперимента при поиске оптимальных условий / Ю. П. Адлер, Е. В. Маркова, Ю. В. Грановский– М.: Наука, 1976. – 280с.

123. Нагорний К. А., Ткачук М. В. Архітектурні моделі оцінки складності пост об’єктна-орієнтованих технологій розробки програмних систем. /М. М. Литвинчук, К. А. Нагорний, М. В. Ткачук // Вісник Харківського національного університету ім.. В.Н. Каразіна. Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – 2012. – № 1015. – с.225–235.

124. Unified Modeling Language (UML) Resource Page //Офіційний вєб-ресурс уніфікованої мови моделювання UML, що розробляється консорціумом OMG – Object Management Group [Електронний ресурс] – Режим доступу: http://www.uml.org

125. OMG Systems Modeling Language //Офіційний вєб-ресурс мови моделювання систем SysML, що розробляється консорціумом OMG – Object

Management Group [Електронний ресурс] – Режим доступу: http://www.omgsysml.org

126. Kogut, P., Clements, P. Features of Architecture Description Languages. / Kogut, P., Clements, P. // Proceedings of Software Technology Conference. – 1995.

127. Garlan, D., Monroe, R., Wile, D. ACME: An Architecture Description Interchange Language / Garlan D., Monroe R., Wile D. // Proceedings of the CASCON '97: conference of the Centre for Advanced Studies on Collaborative research – 1997.

128. Clements, P. A survey of architecture description languages. / Clements P. // Proceedings of the 8-th International Workshop on Software Specification and Design, IEEE Computer Society Press. – 1996. – pp. 16–25.

129. Balmelli, L. An Overview of the Systems Modeling Language for Products and Systems Development / Balmelli L. // IBM Technical Report. – 2006. – 27 p.

130. Vanderperren, Y. SysML and Systems Engineering Applied to UML-Based SoC Design. / Vanderperren Y. // Proceedings of the 2nd UML-SoC Workshop at 42nd DAC – USA. – 2005.

131. Suleiman, B. Non-Functional Property Specifications for Wright ADL / Suleiman B., Tosic V., Aliev E.// School of Computer Science and Engineering, University of New South Wales. – Australia. – 2008. – pp. 766-771.

132. Messina, E. Representation of the RCS Reference Model Architecture Using an Architectural Description Language / Messina E., Dabrowski C. // National Institute of Standards and Technology. Gaithersburg MD, USA. – 2000. – pp. 23-37.

133. Pinto, M. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development / Pinto M., Fuentes L. // University of Malaga – Malaga, Spain. – 2003. – pp. 118-137.

134. Ackbar, J., Batista, T., et al. Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform. // Ackbar J., Batista T., et al. // Proceedings of the WICSA '05: the 5th Working IEEE/IFIP Conference on Software Architecture. – 2005. – 28 p.

135. Goulão, M., Brito, F., Abreu, E. Bridging the gap between Acme and UML 2.0 for CBD. / Goulão M., Brito F., Abreu E. // Proceedings. of the SAVCBS'2003: Specification and Verification of Component-Based Systems Workshop at the ESEC/FSE'2003. – Helsinki, Finland. – September, 2003.

136. Acme ADL Project. Офіційний вєб-ресурс проекту мови опису архітектур ACME [Електронний ресурс] – Режим доступу: http://www.cs.cmu.edu/~acme

137. Pohl, K., Bockle, G., van der Linden, F.J. Software Product Line Engineering. Foundations, Principles and Techniques / Pohl, K., Bockle, G., van der Linden, F.J. – Springer-Verlag, 2005. – 493 p.

138. Apel, S., et al. Model Superimposition in Software Product Lines / Apel S., et al. // Proceedings of the ICMT'2009: International Conference on Model Transformation. – 2009.

139. Figueiredo, E. Concern-Oriented Heuristic Assessment of Design Stability. Submitted for the Degree of PhD. / Figueiredo E. – Lancaster University, Lancaster, UK, October 2009. – 237 p.

140. Marin, M., Moonen, L., Deursen, A.V. A Classification of Crosscutting Concerns. / Marin M., Moonen L., Deursen A.V. // Proceedings of the ICSM'2005: the 21st IEEE International Conference on Software Maintenance, IEEE Computer Society. – 2005. – pp. 673-676.

141. Marin, M., Moonen, L., Deursen, A.V. An Approach to Aspect Refactoring Based on Crosscutting Concerns Types. / Marin M., Moonen L., Deursen A.V. // Proceedings of the MACS'2005: Workshop on Modeling and Analysis of Concerns in Software, ACM. – St. Louis, Missouri. – 2005. – pp. 1-5.

142. Marin, M. Formalizing typical crosscutting concerns. / Marin M. // Proceedings of the 21st IEEE International Conference on Software Maintenance, IEEE Computer Society. – 2005. – pp 673-676.

143. Lanza, M., Marinescu, R. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems/ Lanza, M., Marinescu, R. – Germany: Springer-Verlag, 2006. – 207 p.

144. Ducasse, S., Girba, T., Kuhn, A. Distribution Map. / Ducasse, S., Girba, T., Kuhn, A. // Proceedings of the ICSM'2006: 22nd IEEE International Conference on Software Maintenance, IEEE Computer Society. – 2006. pp. 203-212.

145. Sant'Anna, C., Garcia, A., et al. On the Reuse and Maintenance of Aspect-O riented Software: An Assessment Framework / Sant'Anna C., Garcia A., Chavez C. Lucena C., von Staa A.// Proceedings of the XVII Brazilian Symposium on Software Engineering. – 2003.

146. Eaddy, M., Aho, A., Murphy, G. C. Identifying, Assigning, and Quantifying Crosscutting Concerns / Eaddy M., Aho A., Murphy G. C. // Proceedings of the ACoM '07: the First International Workshop on Assessment of Contemporary Modularization Techniques – 2007.

147. Нагорный К. А. Разработка и применение методики оценки эффективности пост объектно-ориентированных технологий. / Нагорный К.А. // Східно-Европейський журнал передових технологій. – 2013. – № 3/10 (63). – с. 21–25.

148. Сергиенко А. М. Алгоритмические модели обработки потоков данных / А.М. Сергиенко, В. П. Симоненко // Электронное моделирование. – 2008. – Т. 30, №6. – С. 49–60.

149. Ткачук М. В. Модель та інформаційна технологія побудови адаптивної матриці трасування вимог в гнучких процесах розробки програмного забезпечення / М.В.Ткачук, Р.О.Гамзаев // Вісник НТУ «ХПІ». – Харків, 2013. – № 2 (976). – С. 49 – 60.

150. Tkachuk, M. Models, Methods and Tools for Effectiveness Estimation of Post Object-Oriented Technologies in Software Maintenance / M. Tkachuk, K. Nagorniy and R. Gamzayev // V. Yakovyna et al. (Eds.): ICTERI 2015: Revised

Selected Papers, Series title: Communications in Computer and Information Science, Vol. 594: Springer-Verlag Berlin Heidelberg, 2016. – pp. 20-37.

151. Дюк В. Обработка данных на ПК в примерах / Дюк В. – СПб.: Питер, 1997. – 240 с.

152. Дейвисон, М.Л. Многомерное шкалирование / Дейвисон М.Л. – М.: Финансы и статистика, 1988. – 254 с.

153. Ткачук, Н.В. Методика оценки динамической сложности требований в процессах сопровождения унаследованных программных систем / Н. В. Ткачук, К. А. Нагорный, И. О. Мартинкус // Вісник Національного технічного університету "ХПІ" – Харків: НТУ «ХПІ». – 2010. – № 67. – с. 116-125.

154. Gorban, A.N., Zinovyev, A.Y. Principal Graphs and Manifolds. Ch. 2 in: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques / Gorban, A.N., Zinovyev, A.Y.; Olivas E.S. et al. (eds). – USA: Information Science Reference, 2010. – pp. 28-59.

155. Саати, Т.Л. Принятие решений. Метод анализа иерархий / Саати Т.Л.; пер. с англ. – М.: Радио и связь, 1993. – 278 с.

156. Tarcísio, G., Filó, S., et al. A Catalogue of Thresholds for Object-Oriented Software Metrics / Tarcísio G., Filó S., Mariza A. Bigonha S., Kecia A., Ferreira M.// Proccedings of SOFTENG2015: The First International Conference on Advances and Trends in Software Engeneering. – 2015.

157. Нейман, Дж., Моргенштерн, О. Теория игр и экономическое поведение. / Нейман, Дж., Моргенштерн, О.; пер. с англ. – М.: Наука, 1970. – 708 с.

158. Tao, Z. et al. Metrics for Accessing Quality of Product Line Architecture / Tao, Z. et al. // Proceedings of the International Conference on Computer Science and Software Engineering.– 2008.

159. Леоненков А.В. Нечеткое моделирование в среде MATLAB и fuzzyTECH / Леоненков А.В. – СПб.: БХВ-Петербург, 2005. – 736 с.

160. Нагорний К. А. Експериментальне дослідження ефективності пост об’єктна-орієнтованих технологій розробки програмних систем. / К. А. Нагорний, М. В. Ткачук, Ю. М. Жадан // Матеріали XXII міжнародної науково-практичної конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров’я», НТУ «ХПІ», Харків, 15-17 жовтня 2014 р. – с. 15.

161. Visual Paradigm [Електронний ресурс] – Режим доступу: http:// http://www.visual-paradigm.com

162. HAProxy [Електронний ресурс] – Режим доступу: http://www.haproxy.org

163. Apache HTTP Server Project [Електронний ресурс] – Режим доступа URL: http://httpd.apache.org, 09.2015.

164. Apache Tomcat [Електронний ресурс] – Режим доступу: http://tomcat.apache.org

165. Oracle [Електронний ресурс] – Режим доступу: http://www.oracle.com/index.html

166. Ткачук М. В., Нагорний К. А., Ананьев В. О. Розробка архітектури CASE-засобу для дослідження пост об’єктно-орієнтованих технологій. /М. В.Ткачук, К. А. Нагорний, В. О. Ананьев// Матеріали XIX Міжн. науково-практ. конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров’я», НТУ «ХПІ», Харків, 1-3 червня 2011 р. – с. 29.

167. Ларман К. Применение UML 2.0 и шаблонов проектирования: практическое руководство. Третье издание / Ларман К.; пер. с англ. – М.: Вильямс, 2013. – 736с.

168. AcmeStudio. Офіційний веб-сайт розробника [Електронний ресурс] – Режим доступу: http://www.cs.cmu.edu/~./acme/AcmeStudio/index.html

169. Function Point Modeler. Офіційний веб-сайт розробника [Електронний ресурс] – Режим доступу: http://www.functionpointmodeler.com

170. Apache Struts. Офіційний веб-сайт проекту [Електронний ресурс] – Режим доступу: https://struts.apache.org

171. Spring Framework. Офіційний веб-сайт проекту [Електронний ресурс] – Режим доступу: http://projects.spring.io/spring-framework

172. Castor. Веб-сайт проекту [Електронний ресурс] – Режим доступу: https://www2.informatik.hu-berlin.de/~xing/Lib/Docs/Castor/index.html

173. Hibernate. Офіційний веб-сайт проекту [Електронний ресурс] – Режим доступу: http://hibernate.org

ДОДАТОК А.
АКТИ ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ

ЗАТВЕРДЖУЮ:

Проректор з наукової роботи
НТУ «Харківський політехнічний інститут»

_____ проф. А.П. Марченко

"14" квітня 2015р.

АКТ

про використання результатів дисертаційної роботи
аспіранта кафедри програмної інженерії та інформаційних технологій управління
Нагорного Костянтина Анатолійовича
«МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ПОСТ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ»

Ми, що нижче підписалися, декан факультету інформатики та управління проф. І.П. Гамаюн, завідувач кафедри програмної інженерії та інформаційних технологій управління проф. М.Д. Годлевський, професор кафедри програмної інженерії та інформаційних технологій управління проф. М.В. Ткачук склали цей акт у тому, що дисертаційна робота Нагорного К.А. виконувалася на кафедрі програмної інженерії та інформаційних технологій управління (ППТУ) у межах держбюджетної теми МОН України «Розробка інтелектуальних моделей та технологій для підвищення ефективності проектування та супроводу складних програмних систем» (ДР № 0111U002288), де здобувач брав участь як співвиконавець окремих етапів.

В результаті виконання дисертаційної роботи Нагорного К.А. були отримані, зокрема, такі результати:

1. Розроблені моделі та методи для визначення ефективності застосування пост об'єктно-орієнтованих технологій (ПООТ) при супроводі успадкованих програмних систем (УПС), які дозволяють кількісно оцінити ступінь ефективності застосування окремих ПООТ залежно від характеристик конкретних УПС.
2. Реалізовано прикладну інформаційну технологію та методику оцінки ефективності застосування ПООТ, що забезпечило можливість провести експериментальне дослідження запропонованого підходу, оцінити працездатність та надати рекомендації щодо його практичного впровадження.

Декан ІФ-факультету проф. І.П. Гамаюн

Зав. кафедрою ППТУ проф. М.Д. Годлевський

Науковий керівник теми ДР № 0111U002288 проф. М.В.Ткачук

Die erste Adresse für Wissen und Bildung.

bit group

Bit media e-learning solution GmbH
& Co KG
Kärntner Straße 311
8054 Graz,
Austria

Tel: +43 (0) 316 / 28 66 60-0
Fax: +43 (0) 316 / 288 66 60-50
E-Mail: office@bitmedia.cc

11.06.2015

To the Vice-rector for scientific
and research issues
at the National Technical University
"Kharkiv Polytechnic Institute"
Prof. Dr. A. Marchenko

CONFIRMATION LETTER

The company Bit media e-learning solution GmbH & Co KG ascertains with this letter that the results achieved within the PhD-thesis activities performed by Mr. Kostiantyn Nagornyi were used in maintenance process for several software projects performed during 2010-2014, especially the following points:

1. The method of assessment of crosscutting functionality initial and residual level in a software system;
2. The method for estimation of implementation costs for a software system for post object-oriented technologies;
3. The approach to decision making support to choose the most effective post object-oriented technology for a target software system.

These approaches actually can be recognized as the useful tools for legacy software systems maintenance process, because they allow to make reasonable decision which post object-oriented technology is more effective for the given software system. As a praxis result of this approach a number of software defects, related to the crosscutting functionality presence, can be reduced more than 30%, and finally to improve the quality of legacy software systems maintenance.

Kind regards,

Managing Director
Walter Khom

bit media
member of bit group
bit media e-solutions GmbH
Kärntner Straße 311, 8054 Graz
Tel: 0316/28660-0 | Fax: DW 50
office@bitmedia.cc

Офіційний бланк компанії Біт медіа

Проректору з наукової роботи
Національного технічного
університету «Харківський
політехнічний інститут»
д.т.н., проф. А. Марченку

Bit media e-learning solution
GmbH & Co KG
вул. Кертнер, 311
8054 Грац,
Австрія
Tel: +43 (0) 316 / 28 66 60-0
Fax: +43 (0) 316 / 288 66 60-50
E-Mail: office@bitmedia.cc

11.06.2015

АКТ ВИКОРИСТАННЯ.

Компанія Bit media e-learning solution GmbH & Co KG цим листом підтверджує, що результати дисертаційної роботи пана Нагорного Костянтина були використані у процесі супроводу декількох програмних систем на протязі 2010-2014 р.р. і, зокрема, це є наступні:

1. Метод для оцінки початкового та залишкового рівня присутності наскрізної функціональності у програмній системі.
2. Метод для оцінки витрат на впровадження пост об'єктно-орієнтовних технологій у програмній системі.
3. Процедура підтримки прийняття рішень відносно вибору найбільш ефективної пост об'єктно-орієнтовної технології для застосування у цільовій програмній системі.

Слід зазначити, що ці результати є корисним інструментарієм для процесу супроводу успадкованих програмних систем, тому що вони надають можливість прийняти обгрунтоване рішення щодо того, яка із пост об'єктно-орієнтовних технологій найбільш ефективна для певної програмної системи. Практичним результатом використання запропонованого підходу є зменшення більше ніж на 30% кількості програмних дефектів, які зумовлені присутністю наскрізної функціональності, що в кінцевому рахунку призводить до загального підвищення якості процесу супроводу програмних систем.

З найкращими побажаннями,
Управляючий директор

Вальтер Ком

Штамп компанії Bit media

КОПІЯ ВІРНА:

Начальник відділу міжнародних зв'язків
Національного технічного університету
«Харківський політехнічний інститут» **О.А. Гончаров**

ЗАТВЕРДЖУЮ

Директор ТОВ «ІНТЕРПАК ІНФОРМАЦІЙНІ СИСТЕМИ»

О. К. Гамзаєв

"16" квітня 2015 р.

М.П. м. Харків

А К Т

про використання результатів дисертаційної роботи Нагорного Костянтина Анатолійовича, виконаної в НТУ «Харківський політехнічний інститут», що подана на здобуття наукового ступеня кандидата технічних наук

Комісія у складі:

голови - Марчука Артема Володимировича, керівника групи супроводу програмних систем, к.т.н.; членів комісії – Сагайдачного Дениса Олексійовича, провідного інженера групи супроводу програмних систем; Бабака Олександра Юрійовича, старшого інженера-розробника програмного забезпечення,

встановила, що у період 2014 – 2015 рр., при виконанні проектів по супроводу програмних систем для автоматизації організаційних процесів та документообігу в мережі навчальних закладів, були використані наступні результати наукових досліджень Нагорного К.А.:

- методи та інструментальні засоби для визначення стану успадкованих програмних компонентів із урахуванням їх структурної складності та ступеня присутності наскрізної функціональності в вихідному програмному коді;
- архітектурну модель та інструментальні засоби для дослідження технологічних особливостей та ефективності застосування аспектно-орієнтованої технології розробки при супроводі та модернізації успадкованих програмних рішень.

Використання цих результатів, отриманих в дисертаційній роботі Нагорного К.А., дозволило суттєво підвищити якість процесу супроводу успадкованих програмних компонентів за рахунок скорочення трудовитрат та зменшення кількості програмних дефектів.

Голова комісії _____ / Марчук А. В. /

Члени комісії _____ / Сагайдачний Д. О. /

_____ / Бабак О.Ю. /

ЗАТВЕРДЖУЮ:
Проректор НТУ
"Харківський політехнічний інститут"
_____ проф. Р.П. Мигущенко
"20" вересня 2015р.

АКТ
про впровадження результатів дисертаційної роботи
аспіранта кафедри програмної інженерії та інформаційних технологій управління
Нагорного Костянтина Анатолійовича
«МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ПОСТ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ТЕХНОЛОГІЙ»

Ми, що нижче підписалися, декан факультету інформатики і управління проф. І.П. Гамаюн, завідувач кафедри програмної інженерії та інформаційних технологій управління проф. М.Д. Годлевський, професор кафедри програмної інженерії та інформаційних технологій управління проф. М.В. Ткачук, доцент кафедри програмної інженерії та інформаційних технологій управління Р.О. Гамзаєв склали наступний акт у тому, що результати дисертаційної роботи Нагорного К.А. впроваджені в навчальний процес на кафедрі програмної інженерії та інформаційних технологій управління (ПІТУ).

Запропоновані у роботі моделі та процедури оцінки структурної складності та динаміки змін вимог до програмного забезпечення (ПЗ) застосовані в лекційному матеріалі до дисциплін «Основи проектування ПЗ», «Аналіз вимог до ПЗ», «Методи та засоби автоматизації процесів життєвого циклу ПЗ».

Компоненти прикладної інформаційної технології для оцінки ефективності застосування пост об'єктно-орієнтованих технологій використовуються в лабораторних практикумах з дисциплін «Методи та засоби автоматизації процесів життєвого циклу ПЗ» та «Компонентні програмні архітектури».

Декан факультету інформатики та управління
НТУ «ХПІ» д.т.н., проф. _____ І.П. Гамаюн

Зав. кафедри ПІТУ
НТУ "ХПІ", д.т.н., проф. _____ М.Д. Годлевський

Професор кафедри ПІТУ
НТУ "ХПІ", д.т.н., проф. _____ М.В. Ткачук

Доцент кафедри ПІТУ
НТУ "ХПІ", к.т.н. _____ Р.О. Гамзаєв

ДОДАТОК Б.
ПРИКЛАДИ ДОКУМЕНТАЦІЇ ПО РОЗРОБЦІ ІНСТРУМЕНТАЛЬНОГО ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ПООТ

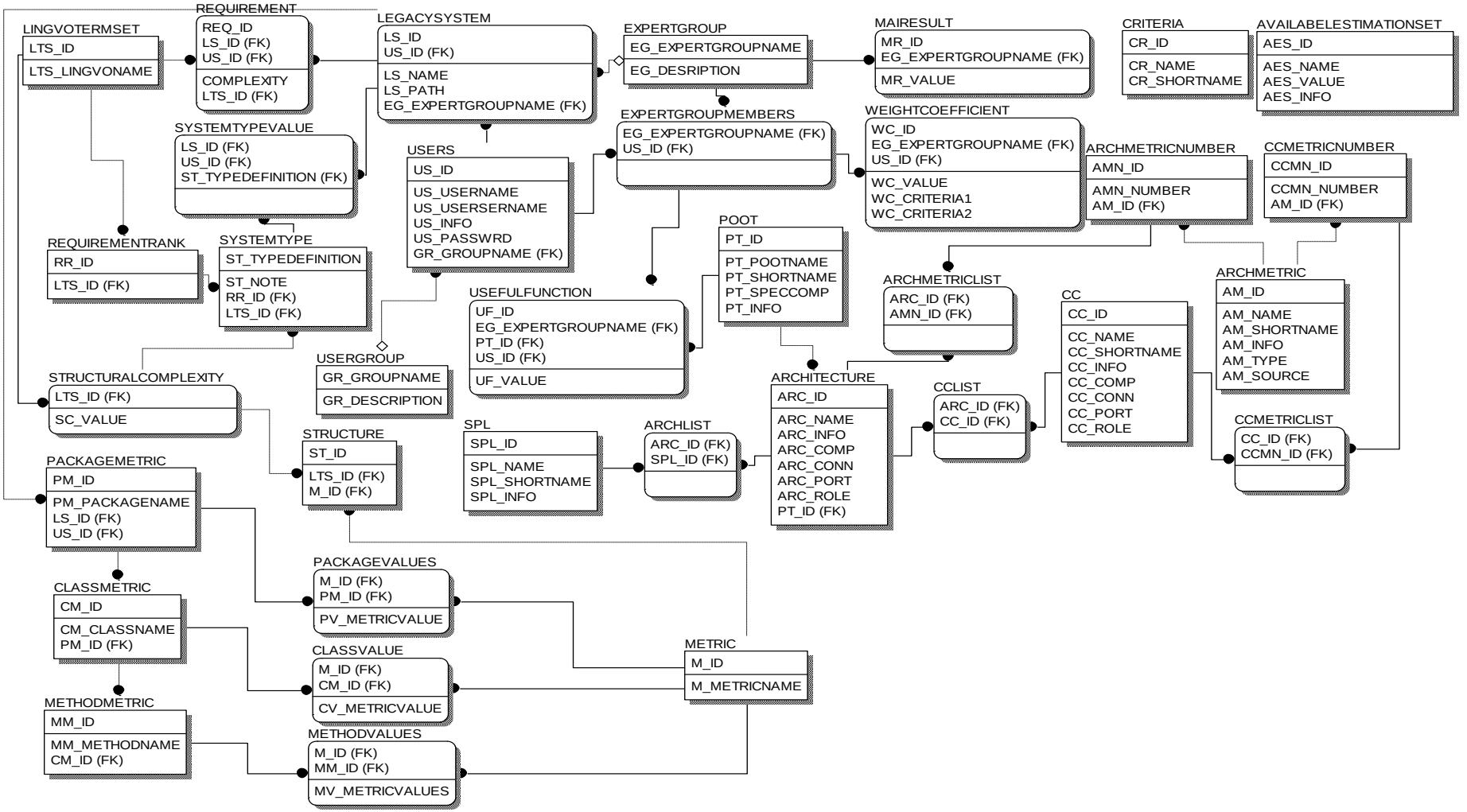


Рисунок Б.1 – Концептуальна модель даних розробленого CASE-засобу

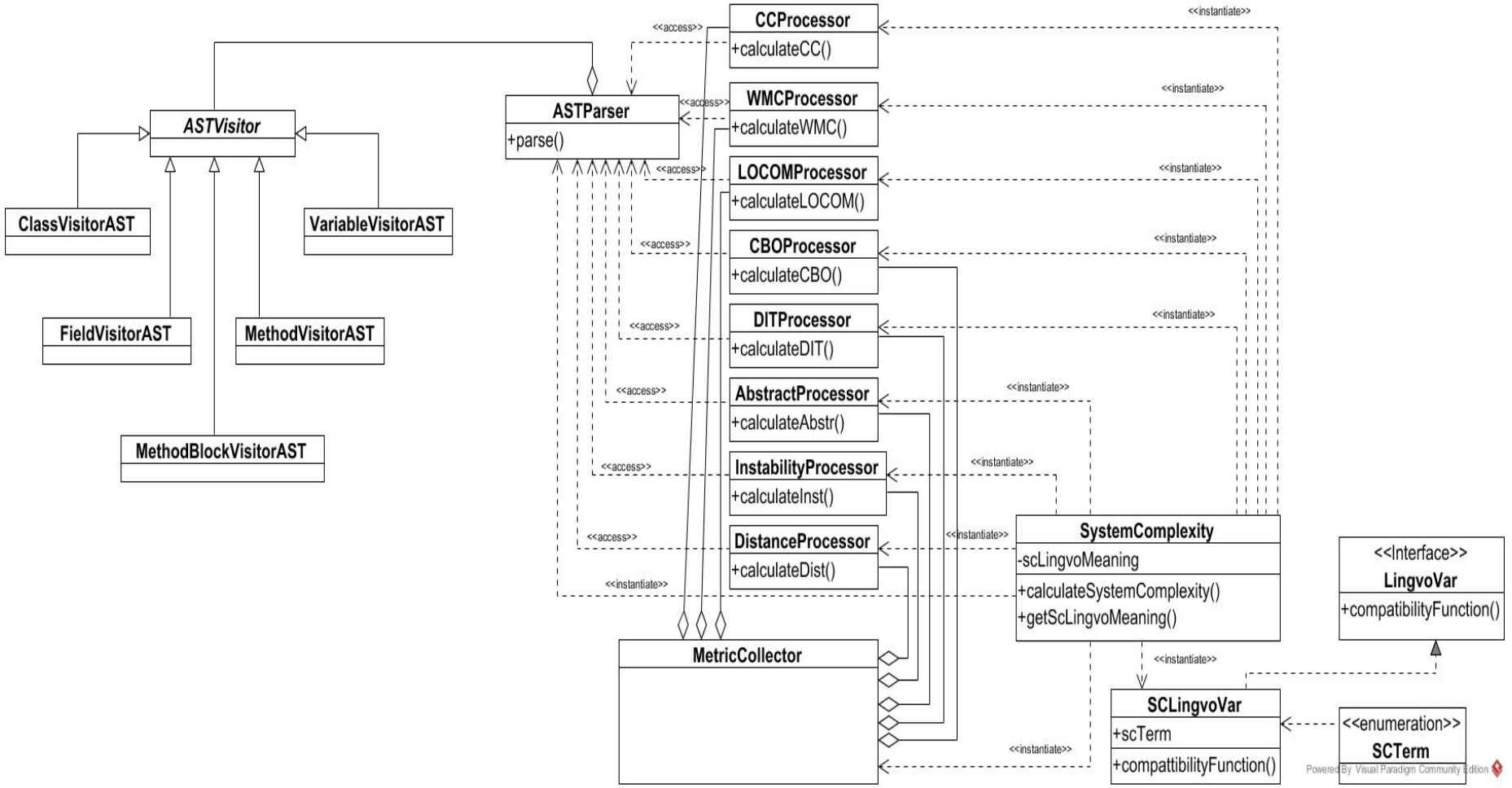


Рисунок Б.2 – Фрагмент діаграми класів розробленого CASE-засобу

для підсистеми визначення структурної складності УПС

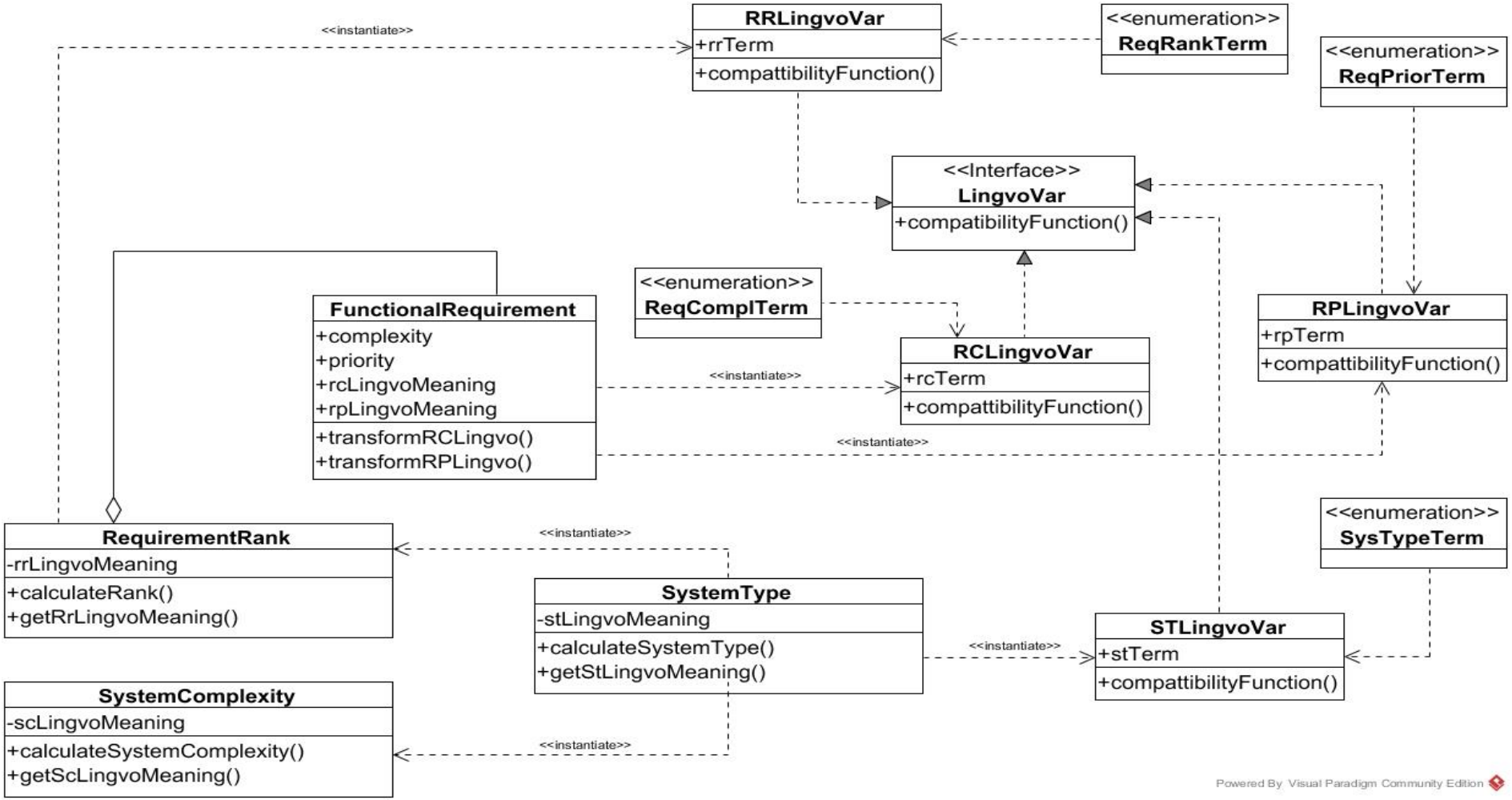


Рисунок Б.3 – Фрагмент діаграми класів розробленого CASE-засобу

для підсистеми визначення типу УПС

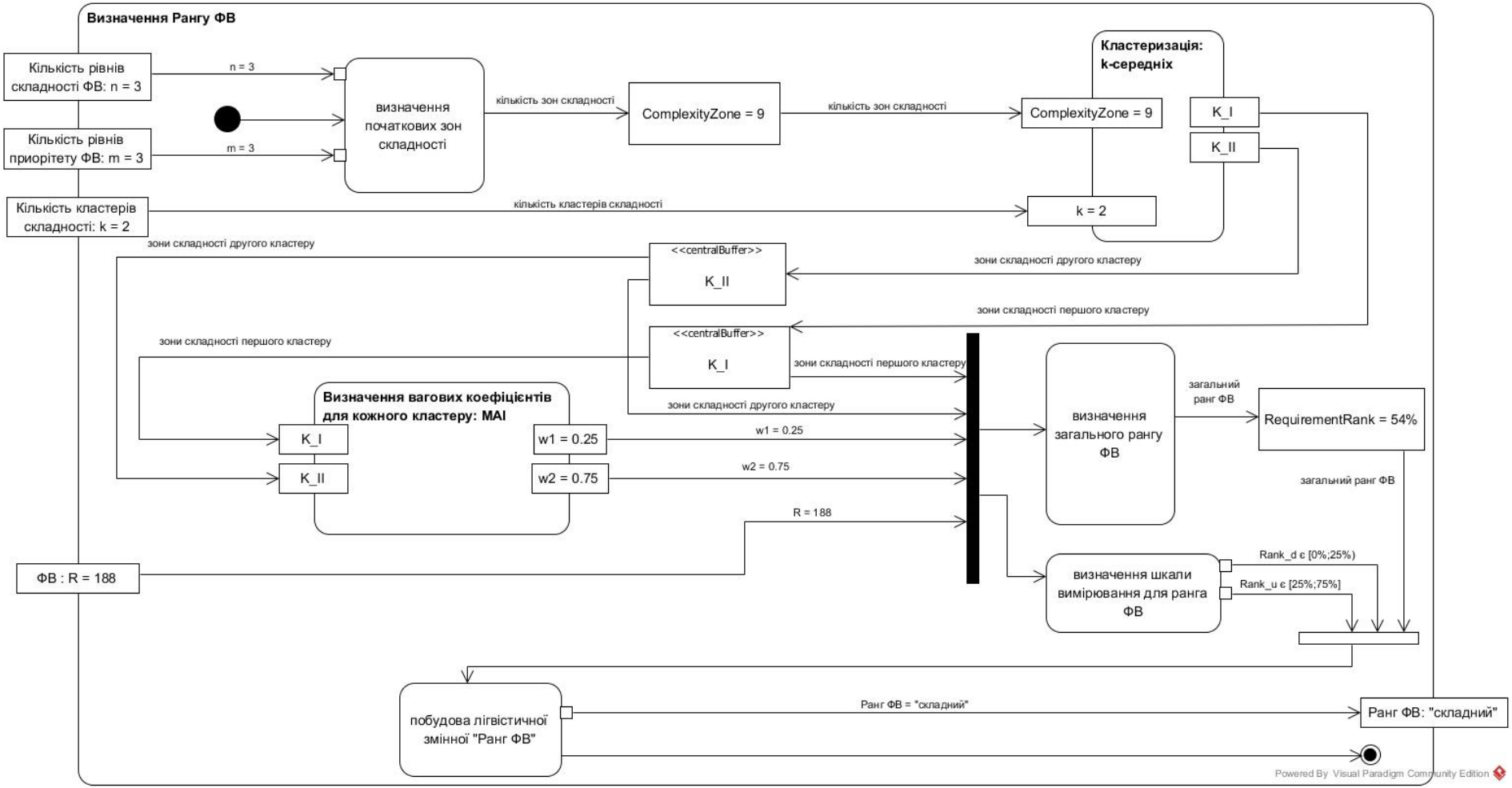


Рисунок Б.4 – Діаграма активностей для визначення загального рангу вимог УПС

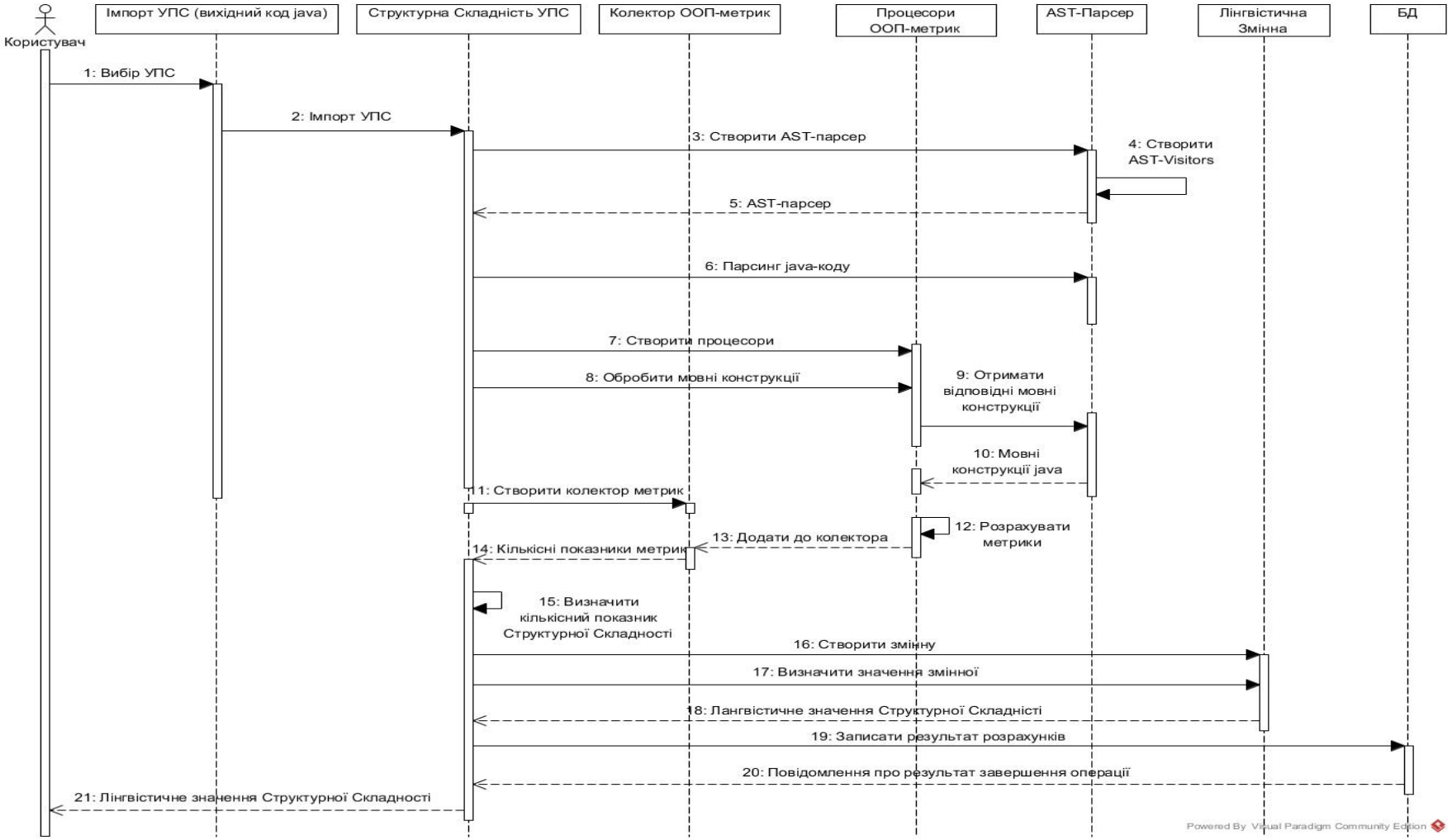


Рисунок Б.5 – Діаграма послідовності для визначення структурної складності УП

ДОДАТКИ