УДК 004.03

Є. В. Левус, М. І. Шалак, О. Я. Вітоль
Національний університет "Львівська політехніка",
кафедра програмного забезпечення

АНАЛІЗ ЕФЕКТИВНОСТІ АСПЕКТНО-ОРІЄНТОВАНОЇ РЕАЛІЗАЦІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ СУПРОВОДУ СИСТЕМИ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

Розглянуто аспектно-орієнтоване програмування як метод інженерії програмного забезпечення для підвищення супроводжуваності клієнт-серверних програмних систем. Наведено результати перепроектування раніше створеної системи на основі об'єктно-орієнтованої технології з метою локалізації наскрізної функціональності. Використано відповідні метрики коду для порівняння ефективності об'єктно-орієнтованої та аспектно-орієнтованої реалізацій цієї системи. Виявлено найбільш та найменш ефективні аспектно-орієнтовані реалізації функціональності клієнт-серверної системи.

Ключові слова: супровід програмного забезпечення, модуль системи, об'єктно-орієнтоване програмування, аспектно-орієнтована реалізація, клас, аспект, наскрізна функціональність, метрика коду, індекс супроводжуваності.

In this article an aspect-oriented programming is considered as a software engineering method for providing maintenance of client-server software systems. The results of redesign system from object-oriented to aspect-oriented by separation of cross-cutting functionality are shown. The relevant code metrics are used to compare object-oriented and aspect-oriented implementations of the system. There have been detected the most and the least effective aspect-oriented implementations of functionality of the client-server system.

Key words: software maintenance, system module, object-oriented programming, aspect-oriented implementation, class, cross-cutting functionality, code metric, maintainability index.

I. Проблема зростання складності супроводу програмного забезпечення

Супровід програмного забезпечення (ПЗ) є невід'ємною частиною життєвого циклу розроблення програмного забезпечення, що триває після передавання його в експлуатацію. Процес супроводу стандартизовано організацією ISO, що задокументовано в ISO/IEC14764. Цей етап життєвого циклу розроблення ПЗ розглядається з погляду задоволення вимог споживача у готовому програмному продукті. Близько 70 % часу у проекті витрачається на супровід і внесення змін у готовий проект [1]. Вартість етапу супроводу оцінюється приблизно в 50 % від загальної вартості життєвого циклу [2].

Сучасний стан інженерії ПЗ характеризується зростанням складності створюваних програмних систем, а разом з тим підвищується складність виконання робіт на етапі супроводу. Складність програмної системи в інженерії ПЗ трактується як на інтуїтивно зрозумілому рівні, так і з використанням вимірювальних характеристик – метрик ПЗ. Загалом складність ПЗ визначається кількістю його складових частин, зв'язків між ними, що відображають взаємодію в системі. У роботі [3] досліджено, як складність програми впливає на вартість супроводу. Загалом для складних програм ця вартість буде на 25 % вищою, ніж для супроводу простих програм. Це зумовлено тим, що у разі підтримки великих програмних рішень з часом ускладнюється знаходження та виправлення помилок, додавання нових функціоналів та інших характеристик, оскільки стає все важче визначити, як зміни вплинуть на систему, чи не внесуть додаткові помилки та дефекти.

Складність супроводу у такому випадку настільки велика, що це зумовлює розгляд його складових задач як окремих проектів розроблення ПЗ.

Управління складністю ПЗ виконується на основі поняття модуля, складової частини програмної системи. Розвиток поняття модуля зумовлював створення нових методів інженерії ПЗ (рис. 1).

Рис. 1. Значення модуля в інженерії ПЗ

Об'єктно-орієнтоване програмування (ООП) є основним методом інженерії ПЗ. З погляду забезпечення супроводу програмних систем цей метод має недоліки, а саме: неможливість локалізації наскрізної функціональності в одному класі [4]. Використовуючи ООП, важко забезпечити вставки і модифікацію коду для додавання нової функціональності у вже створену систему так, щоб не допустити внесення помилок. Окрім цього, модифікація кожної частини системи окремо під нові вимоги може призвести до несумісності.

У великих проектах додавання функціонала у ООП-проект призводить до виникнення таких проблем [5]:

1) збільшення зв'язності класів;
2) збільшення глибини дерева наслідування для ієрархії класів;
3) недостатня модульність.

Всі ці проблеми свідчать про те, що буде складно вносити зміни в програмний код і використовувати його повторно. Причиною є наявність наскрізної функціональності в різних модулях системи, що ускладнює інспектування коду, виявлення помилок, значно ускладнює супровід.

Отже, існує проблема зростання складності супроводу ПЗ. Актуальним є пошук рішення для програмного відокремлення наскрізної функціональності від основної логіки програми, що зменшить складність програмної системи, знизить вартість та важкість її супроводу.

II. АОП для вирішення проблеми зростання складності ПЗ

Одним із способів вирішення проблеми наскрізної функціональності є використання пост-об'єктно-орієнтованих технологій (ПООТ) [6]. До найвідоміших ПООТ можна зарахувати: аспектно-орієнтовану технологію (aspect-oriented technology), властивості-орієнтовану технологію (feature-oriented technology) і контекстно-орієнтовану технологію (context-oriented technology). Аспектно-орієнтоване програмування – це нова парадигма програмування, яку запропонувала група інженерів дослідницького центру Xerox PARC під керівництвом Грегора Кічалеса. Завдання АОП – виділення наскрізної функціональності у спеціальні модулі – аспекти (рис. 2).

Класи ООП, які містять і основну логіку програми, і наскрізну функціональність, відображені ліворуч. АОП дає змогу залишити в класах лише основну функціональність, а наскрізна функціональність виноситься у аспекти.

У роботах [6–9] доведено ефективність застосування ПООТ. Зокрема, в [6] відображено, що ООП недостатньо для вирішення проблеми наскрізної функціональності (ефективність всього 6,7 %), а найкращим рішенням є аспектно-орієнтоване програмування, результат використання якого забезпечив понад 70 % ефективності.

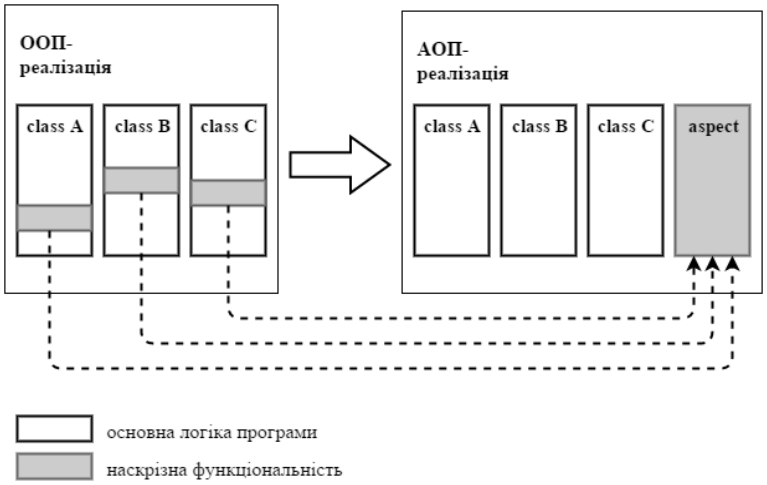

Рис. 2. Основна ідея АОП

У роботах [10–11] проведено порівняння ефективності застосування аспектно-орієнтованого підходу з об'єктно-орієнтованим для окремого модуля захисту розподіленої системи теплового проектування за низкою метрик. Показано, що використання АОП покращує надійність програмного забезпечення за рахунок меншої складності проекту.

Розробляючи програмне забезпечення з використанням АОП, спочатку необхідно виконати декомпозицію функціональностей системи на функціональність модульного рівня та наскрізну функціональність, далі – останню реалізувати за допомогою аспектів. Кінцевий продукт отримують, вплітаючи аспекти (aspect weaving) [10].

Структуру аспекту подано на рис. 3.

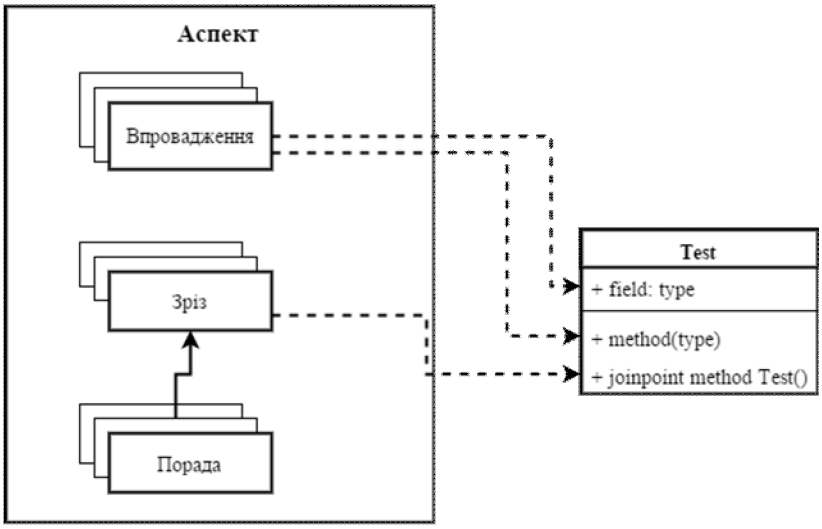

Рис. 3. Структура аспекту

Порада (advice) – код, який виконається до, після чи замість точки з'єднання (joinpoint), якою може бути виклик методу Test() в класі Test. Зріз (pointcut) задає список точок з'єднання. Впровадження (introduction) – в АОП є можливість визначати в аспектах нові поля і методи, які будуть впроваджені в ООП-класи.

За правильного використання АОП дає змогу: покращити декомпозицію системи, забезпечити повторне використання коду [12].

Оскільки аспектно-орієнтована парадигма є доволі молодою, актуально проаналізувати використання АОП для розроблення систем з погляду супроводжуваності коду, підтвердження доцільності його застосування до тих чи інших класів задач, пошуку засобів підтвердження ефективності використання цього підходу в кожному конкретному випадку[13].

III. Постановка задачі до експериментів

Для аналізу ефективності застосування аспектно-орієнтованого підходу пропонується:

- Проаналізувати раніше створену систему на наявність наскрізної функціональності.
- Локалізувати в аспектах наскрізну функціональність із використанням АОП.
- Обчислити індекс супроводжуваності для реалізації класів системи з та без використання аспектів.

Для обчислювальних експериментів взято систему е-Куратор, яка призначена для роботи користувачів з трьома рівнями доступу в мережі [14].

У системі виділено таку наскрізну функціональність:

1. Обробка виняткових ситуацій, насамперед проблеми доступу до бази даних. Для повідомлення користувача про помилку використовується вікно, яке випливає. Без використання АОП довелося б обробку виняткової ситуації здійснювати у кожному місці, де відбувається звернення до бази даних, із викликом відповідного повідомлення. Далі наведено фрагмент аспекту для обробки виняткових ситуацій:

```
    @Around("execution(public * com.ecurator.dao..*(..) throws SQLException)")
трьох
    public Object errorHandling(ProceedingJoinPoint joinPoint) throws
Throwable{
    try {
     //Продовжити виконання при відсутності помилок
           return joinPoint.proceed();
     } catch (SQLException exception) {
     //Вивести повідомлення про помилку роботи з базою даних
     Messagebox.show("Помилка доступу до бази даних", "Помилка",
Messagebox.YES, Messagebox.ERROR);
     }

     return new ArrayList<Object>();
    }
```

2. Ведення журналу подій, що дає змогу відстежувати, який метод викликається і з якими аргументами, що згодом спрощує виявлення місця, де сталася помилка. Застосування аспектів у цьому випадку є одним з найефективніших, оскільки істотно зменшує кількість коду в класах з бізнес-логікою програми, виносячи її в аспекти. З використанням порад навколо виконуваних методів можна отримати дані про виклик методу та повернене значення.

3. Перевірка прав доступу за допомогою винесення коду перевірки в окремий аспект. Для позначення рівня прав (студент, куратор, кафедра) використовуються анотації, що й згодом обробляються в аспекті. Анотується окремий метод, виконання якого потребує спеціальних прав. Аспект обробляє лише методи, для яких наявні анотації типу Security, згодом зчитує з них рівень доступу. Для відповідної перевірки без АОП потрібно застосовувати умовний оператор, тому використання АОП робить перевірку компактнішою.

4. Відкриття та закриття з'єднання з базою даних. Проводиться в методах доступу до даних. Для використання аспектів необхідно виконати зріз по всіх методах пакета доступу до бази даних. Перед виконанням методу з'єднання відкривається, метод отримує дані, згодом з'єднання закривається і дані повертаються на відповідний запит.

5. Ініціалізація закритих атрибутів класу. У проекті використовується компонентний каркас ZK, що дає змогу описувати елементи графічного інтерфейсу користувача за допомогою об'єктно-орієнтованого коду. Проте, як і в будь-яких бібліотеках, в ньому є певні недоліки, що проявляються у разі впровадження в конкретну систему. Наприклад, щоб отримати доступ до об'єкта, що представляє елемент графічного інтерфейсу і міститься всередині контейнера, необхідно викликати метод getFellow у методі класу для кожного елемента. Це збільшує обсяг коду та може призвести до помилок, якщо пропущено один з елементів. Навіть якщо оформити ініціалізацію атрибутів класу, використовуючи мовні засоби Java з виділенням допоміжного класу, необхідно викликати метод

цього класу перед зверненням до атрибутів. Це знову відволікатиме увагу розробника та може призвести до помилок. Тому доцільно вирішити цю проблему за допомогою аспекту, що викликатиме метод getFellow для кожного атрибута перед виконанням методу onCompose.

```
//Наскрізна функціональність для ініціалізації елементів інтерфейсу
public void afterCompose() {
 //Наповнення об'єктів графічного інтерфейсу початковими значеннями із розмітки сторінки
  addBtn = (Button) getFellow("addBtn");
  groupsBtn = (Button) getFellow("groupsBtn");
  usersGrid = (Grid) getFellow("usersGrid");
  footer_msg = (Footer) getFellow("footer_msg");
}
```

IV. Результати застосування АОП

Комплексною оцінкою характеристик, що визначають здатність програми до розвитку, є індекс супроводжуваності (MI). Індекс супроводжуваності розробили в університеті штату Айдахо в 1991 р. вчені Оман і Гагемейстер [15]. Формула оцінки сформувалася внаслідок аналізу статистичних даних великої кількості проектів, її відкоригували багато інженерів та команди підтримки програмних систем. Вона використовувалась у компанії Hewlett Packard для вибору програмного забезпечення з-поміж конкуруючих продуктів. Метрика MI інтегрована в середовище розробки Visual Studio. Формула для обчислення індексу супроводжуваності [16] має вигляд:

$$MI = 171 - 5,2\ln(HV) - 0,23CC - 16,2\ln(LOC), \quad (1)$$

де HV – метрика Холстеда – об'єм Холстеда; CC – цикломатична складність; LOC – кількість рядків коду.

Кількість рядків вихідного тексту і метрика Холстеда належать до метрик розміру програм. Вони аналізують вихідний текст програми і використовуються для оцінювання складності проміжних продуктів розробки.

Метрика Холстеда обчислюється на основі аналізу кількості рядків та синтаксичних елементів вихідного коду програми. Основа метрики Холстеда – чотири вимірювані характеристики програми: кількість унікальних операторів програми, враховуючи символи-роздільники, імена процедур і знаки операцій (словник операторів); кількість унікальних операндів програми (словник операндів); загальна кількість операторів у програмі; загальна кількість операндів у програмі.

Об'єм Холстеда визначається за формулою [16]:

$$HV = N\log_2 n \quad (2)$$

Цикломатична складність коду створюється обчисленням кількості різних шляхів коду в потоці програми. Програма зі складним потоком управління потребує ретельнішого тестування для досягнення прийнятного рівня покриття коду і характеризується нижчою супроводжуваністю.

Цикломатична складність розраховується так [17]:

$$M = E - N + P + R, \quad (3)$$

де M – цикломатична складність; E – кількість вершин графу; N – кількість ребер графу; P – кількість з'єднаних компонентів; R – кількість зворотних тверджень.

Показник цикломатичної складності розраховується для модуля, методу та інших структурних одиниць програми [18].

Що вищі значення цих метрик, то гіршою є система з погляду топологічної складності.

Згідно з рекомендаціями фірми Microsoft значення метрики MI трактується так [19]:

0–9 – код важко підтримувати;

10–19 – посередній показник легкості підтримки;

20–100 – код легко підтримувати.

Тобто збільшення значення цієї метрики свідчить про якісніший код у плані підтримки [20].

У разі впровадження аспектно-орієнтованого підходу впливу зазнаватимуть різні компоненти індексу супроводжуваності, як з позитивного, так і з негативного боку. Тому важливо знайти оптимальний баланс цих показників для забезпечення відповідного рівня супроводжуваності.

Для розв'язання поставлених задач використано середовище розробки Eclipse, мову Java та бібліотеку AspectJ. Ці програмні засоби добре інтегруються, із AspectJ постачається застосунок до середовища, що дає змогу відстежувати методи та класи, в які будуть внесені зміни аспектів.

Для обчислення метрик коду використаний додаток CodePro, що містить підтримку всіх необхідних вимірювань, що входять в індекс супроводжуваності, а також засоби візуалізації обчислень та пояснення значень кожної з метрик.

Під час порівняння аспектно-орієнтованої та об'єктно-орієнтованої реалізацій клієнт-серверної системи за індексом супроводжуваності отримано результати, відображені у табл. 1.

Таблиця 1

Значення індексу супроводжуваності

Наскрізна функціональність	MI для реалізації за допомогою ООП	MI для реалізації за допомогою АОП	Зміна індексу супроводжуваності MI, %
Обробка виняткових ситуацій	**44.62**	**47**	**5**
Ведення журналу подій	**51,90**	**54,21**	**4,5**
Перевірка прав доступу	**45,33**	**44,02**	**-3**
Відкриття та закриття з'єднання з базою даних	**46,68**	**50,12**	**7**
Ініціалізація закритих атрибутів класу	**46.68**	**50.12**	**7**

Під час обчислення покращення значення метрик ООП-реалізації прийнято за 100 %, значення метрик АОП реалізацій відображено у відсотках відносно ООП реалізації.

Після використання аспектів для обробки виняткових ситуацій роботи з базою даних та внесення змін у всі класи, де такі ситуації обробляються (виведенням повідомлення на екран), отримано результати, відображені на рис. 4.

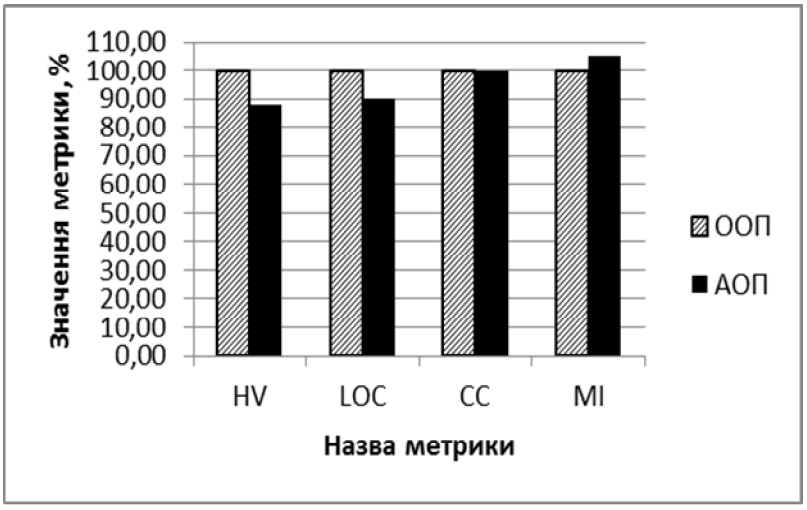

Рис. 4. Зіставлення метрик у АОП і ООП реалізаціях для обробки виняткових ситуацій

Зростання індексу супроводжуваності становило 5 %. При цьому метрика Холстеда та кількість рядків коду суттєво зменшилися, що позитивно впливає на систему.

Для відслідковування доцільності внесення змін за допомогою аспектів проаналізовано значення індексу супроводжуваності після модифікації кожного наступного класу (рис. 5).

Рис. 5. Відображення значень MI внаслідок внесення змін у більшу кількість класів

Отже, за такої реалізації аспекту та класів доцільно впроваджувати АОП з можливістю модифікації трьох класів і більше. Інакше навіть в разі покращення якості коду класів супроводжуваність системи буде погіршуватись за рахунок введення додаткового коду аспекту.

Ефективнішим виявилося впровадження аспектно-орієнтованого підходу для ініціалізації приватних атрибутів класів-компонент, що представляють графічний інтерфейс користувача. Така модифікація дозволила вилучити значну кількість однорідного коду, прив'язавши ідентифікатори графічних компонент до назв атрибутів класу.

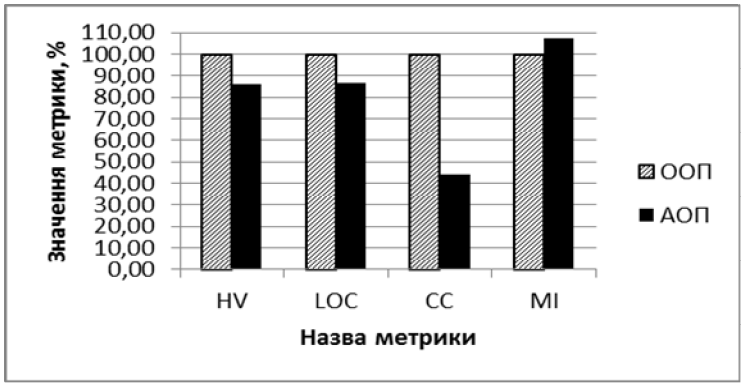

Рис. 6. Зіставлення метрик у АОП- і ООП-реалізаціях для ініціалізації приватних атрибутів

Зростання індексу супроводжуваності становило 7 % (рис. 6). Всі три метрики топологічної складності суттєво зменшилися. Варто звернути увагу на те, що значення цикломатичної складності зменшилось більш ніж на 50 %.

Для ведення журналу подій відбувався запис у файл під час виклику і завершення виконання кожного методу, зокрема його параметри, назва класу та значення, що повертається. В результаті покращення супроводжуваності зміненого коду досягло 4,5 % порівняно із реалізацією без аспектів (рис. 7). При цьому значення метрики Холстеда зменшилось на 25 %, також вдалося знизити кількість рядків коду.

У відсотковому співвідношенні найістотніше вдалося покращити код для роботи з базою даних, де потрібно відкривати та закривати з'єднання під час виконання запиту. Такий код був винесений в аспект.

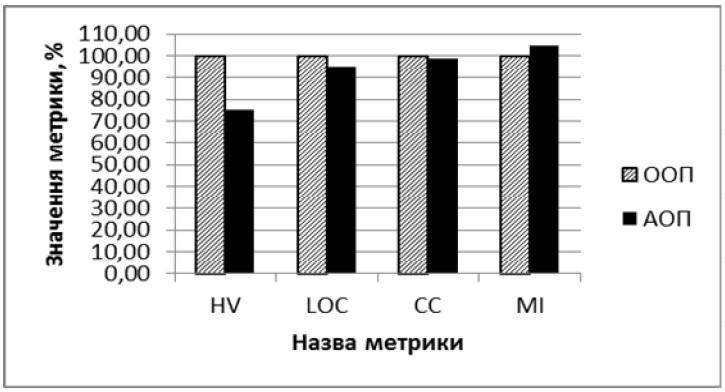

Рис. 7. Зіставлення метрик у АОП- і ООП-реалізаціях для ведення журналу подій

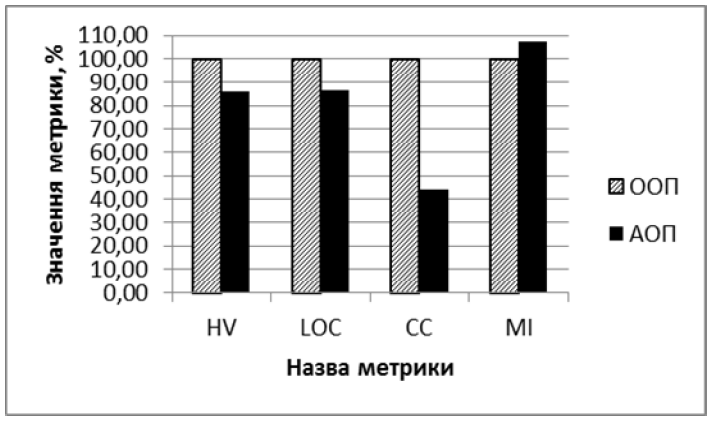

Рис. 8. Зіставлення метрик у АОП- і ООП-реалізаціях для роботи з базою даних

Покращення коду внаслідок впровадження АОП в пакет для роботи з базою даних досягло 7 %, значення метрик топологічної складності істотно зменшилися (рис. 8). Особливо варто звернути увагу на те, що значення метрики цикломатичної складності знизилось більш ніж на 50 %.

Негативногой результату досягнуто для перевірки прав доступу – зниження індексу супроводжуваності на 3 % (рис. 9). Пов'язано це з необхідністю винесення коду із особливим доступом в окремий метод для застосування анотацій, що призвело до збільшення кількості рядків коду.

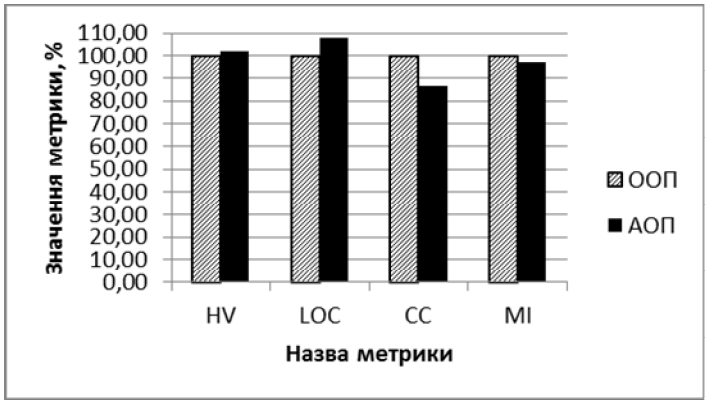

Рис. 9. Зіставлення метрик у АОП- і ООП-реалізаціях для перевірки прав доступу

Проте таке впровадження можна розглядати як позитивне для кращого розуміння коду, оскільки застосування анотацій простіше, ніж використання кожного разу умовного оператора –

кількість символів у рядку коду зменшується. Також досягнуто зниження значення метрики цикломатичної складності на більш ніж 10 %.

Висновки

У роботі наведено результати порівняння застосування ООП та АОП для реалізації системи клієнт-серверної архітектури. Аналіз ефективності АО-реалізації виконано з використанням метрик ПЗ. Результати свідчать про зростання індексу супроводжуваності системи після застосування АОП, а саме: від 4 % до 10 % для змінених ділянок коду. Отримано негативний результат індексу супроводжуваності для перевірки прав доступу. Це пов'язано з необхідністю винесення коду із особливим доступом в окремий метод для застосування анотацій, що призвело до збільшення кількості рядків коду. У конкретній реалізації систем найефективнішим відносно загальної кількості рядків коду виявилось застосування АОП для роботи з базою даних, найменш ефективним – для перевірки прав доступу. Доцільно впроваджувати АОП з можливістю модифікації трьох класів і більше. Інакше навіть у разі покращення якості коду класів супроводжуваність системи буде погіршуватись за рахунок введення додаткового коду аспекту. Перспективним є подальше дослідження впливу АОП на супроводжуваність у разі застосування його для інших підсистем, використання інших метрик та огляд стосовно інших вимог до системи, таких як ефективність використання ресурсів, час виконання тощо. Також перспективним є дослідження впливу АОП на супроводжуваність у разі його застосування у реалізації шаблонів проектування.

1. Zelkowitz M. V. Principles of Software Engineering and Design / Zelkowitz M. V., Shaw A. C., Gannon J. D. // Prentice-Hall, Inc., Englewood Cliffs, NJ. – 1979. 2. VAN Vliet H. Software Engineering: Principles and Practices / 2nd Edition. John Wiley & Sons, West Sussex, England, – 2000. 3. Rajiv D. Banker. Software complexity and maintenance costs / Srikant M. Datq Chris F. Kemerer, Dani Zweig // Communications of the ACM. – 2003. – № 11. – P. 81–94. 4. Kiczales G., Aspect-oriented programming / Kiczales G., Lamping J., Mendhekar A. – Springer-Verlag: Palo Alto Research Center, 1997. – 41 p. 5. Ткачук Н. В. Об одном подходе к оценке эффективности применения постобъектно-ориентированных технологий при сопровождении программных систем / Ткачук Н. В., Нагорный К. А. // Проблемы программирования (Problems in Programming. Scientific Journal). ISSN 1727 – 4907 – К.: НАН Украни. – 2010. – № 2–3 (спец. выпуск). – С. 252–260. 6. Tkachuk Mykola. Knowledge-Based Approach to Effectiveness Estimation of Post Object-Oriented Technologies in Software Maintenance / Mykola Tkachuk, Konstiantyn Nagornyi, Rustam Gamzayev. // Proceedings of the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, Lviv, Ukraine, May 14–16, 2015. 7. Нагорный К. А. Разработка и применение методики оценки эффективности постобьектно-ориентированных технологий / К. А. Нагорный // Восточно-Европейский журнал передовых технологий. – 2013. – № 10. – С. 23–28. 8. Ткачук Н. В. Об одном подходе к оценке эффективности применения постобъектно-ориентированных технологий при сопровождении программных систем / Н. В. Ткачук, К. А. Нагорный // Проблемы программирования (Problems in Programming. Scientific Journal). ISSN 1727 – 4907 – К.: НАН Украни. – 2010. – № 2–3 (спец. выпуск). – С. 252–260. 9. Нагорний К. А. Архітектурні моделі та метрики оцінювання складності застосування постоб'єктно-орієнтованих технологій розробки програмних систем / М. М. Литвинчук, К. А. Нагорний, М. В. Ткачук // Вісник ХНУ ім. В. Н. Каразіна, Серія "Математичне моделювання. Інформаційні технології. Автоматизовані системи управління". – 2012. – № 1015. – С.234–245. 10. Яковина В. С. Аналіз використання аспектно-орієнтованого програмування як засобу підвищення надійності програмного забезпечення / В. С. Яковина, Д. В. Федасюк, Н. М. Мамроха. // Інженерія програмного забезпечення. – 2010. – № 2. – С. 23–28. 11. Яковина В. С. Аспектна декомпозиція компонентів захисту розподіленої системи теплового проектування / В. С. Яковина, Н. М Мамроха., М. М. Сенів // Збірник матеріалів шостої міжнародної конференції "Інтернет-Освіта-Наука-2008" ІОН-2008. – Вінниця. – Т. 2. – С. 407–410. 12. Gary Pollice. Professor of Practice, Worcester Polytechnic Institute. Aspect-Oriented Programming: What is it good for [Електронний ресурс]. – Режим доступу: http://www.ibm.com/developerworks/rational/library/mar06/pollice/. 13. Friedrich Steimann, The

Paradoxical Success of Aspect-Oriented Programming / Friedrich Steimann – Hagen: Lehrgebiet Programmiersysteme, 2006. – 28 p. 14. Шалак М. Застосування аспектно-орієнтованого підходу в інженерії програмних систем, що вимагають тривалого супроводу / М. Шалак, Є. Левус // 70–та студентська науково-технічна конференція : збірник тез доповідей / Національний університет "Львівська політехніка". – Львів : Вид-во Львівської політехніки, 2012. – С. 163. 15. Oman P. W, A Definition and Taxonomy for Software Maintainability / Hagemeister J., Ash D. – Technical Report #91–08–TR, Software Engineering Test Laboratory, University of Idaho, Moscow, 1991. 16. Introduction to Code Metrics. [Електронний ресурс]. – Режим доступу : http://radon. readthedocs.org/en/latest/intro.html. 17. Безменов М. І. Метрики як оцінка моделей якості програмного забезпечення медичного обладнання / М. І. Безменов, О. М. Ланських, В. Г. Борисов // Вестник Национального технического университета "ХПИ" : Системный анализ, управление и информационные технологии. – 2010. – № 9. – С. 188–196. 18. Поморова О. В. Аналіз методів та засобів оцінки якості програмних систем / О. В. Поморова, Т. О. Говорущенко // Радіоелектронні і комп'ютерні системи. – 2009. – № 6. – С. 148–158. 19. Code Metrics Values [Електронний ресурс] / Microsoft – 2012. – Режим доступу: http://msdn.microsoft.com/en-us/library/ bb385914.aspx. 20. Метрики кода и их практическая реализация в Subversion и ClearCase. Часть 1 – метрики [Електронний ресурс] / Александр Новичков, Александр Шамрай. – 2012. – Режим доступу: http://cmcons.com /articles/CC_CQ/dev_metrics.

УДК 004.89

В. В. Литвин, М. Я. Гопяк, О. В. Оборська, Р. В. Вовнянка
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

МЕТОД ПОБУДОВИ ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ НА ОСНОВІ АДАПТИВНИХ ОНТОЛОГІЙ

Розглянуто метод побудови інтелектуальних агентів з використанням онтологічного підходу. Здійснено класифікацію таких агентів з погляду їх функціонування на основі онтологій. Розроблено математичне забезпечення функціонування інтелектуальних агентів, яке ґрунтується на адаптивних онтологіях. Модель адаптивної онтології визначено як розвиток класичної моделі онтології додаванням ваг важливості понять та відношень, які зберігаються в онтології.

Ключові слова: адаптивна онтологія, база знань, інтелектуальний агент, ваги важливості понять та відношень.

In the article the problem of building intelligent agent whose knowledge base core is ontology has been solved. Classification of those systems according to their functioning has been done. For each class appropriate mathematical software has been developed. Intelligent agent models which functioning is based on the ontology has been investigated. The concept of adaptive ontology has been introduced. The model of adaptive ontology is considered as development of the classic model by adding importance weights of the concepts and relations that are stored in the ontology.

Key words: adaptive ontology, knowledge base, intelligent agent, the weight of importance of concepts and relationships.

Вступ. Загальна постановка проблеми

Для побудови інтелектуальних агентів (ІА) використовують інформаційні технології (ІТ). ІТ трактують як певну точку в просторі чотирьох інженерій (комп'ютерної, програмної, системної,