УДК 004.412.2:004.413.5

В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха

АНАЛІЗ ВИКОРИСТАННЯ АСПЕКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ ЯК ЗАСОБУ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Національний університет "Львівська політехніка"
fedasyuk@lp.edu.ua

Проведено порівняння ефективності застосування аспектно-орієнтованого підходу з об'єктно-орієнтованим за рядом метрик. Показано, що використання цієї технології покращує надійність програмного забезпечення за рахунок меншої складності проекту.

Проведено сравнение эффективности применения аспектно-ориентированного подхода с объектно-ориентированным по ряду метрик. Показано, что использование данной технологии улучшает надежность программного обеспечения за счет меньшей сложности проекта.

The comparison of aspect-oriented paradigm with object-oriented one has been carried out using a set of metrics. It was shown that using of this technique improves the software reliability due to lesser project complexity.

Ключові слова: аспектно-орієнтоване програмування, метрики програмного коду, надійність програмного забезпечення

Вступ

На сучасному етапі розвитку програмної інженерії більшості програмних систем властива велика складність. Внаслідок цього, один розробник не в змозі повністю осягнути усю систему і всі її деталі. Високий рівень складності системи є неминучим явищем, яке не можна повністю подолати, а лише частково усунути.

Складність програмних систем обумовлена чотирма основними причинами: складністю предметної області; важкістю управління процесом розробки; необхідністю забезпечити достатню гнучкість програми; незадовільними способами опису поведінки великих дискретних систем [1].

Для вирішення проблеми розробки, модернізації та супроводу складних програмних систем можна використати нову технологію під назвою АОП – аспектно-орієнтоване програмування [2]. Ця технологія – це методика програмування, що базується на понятті аспекту – блоку коду, що інкапсулює наскрізну поведінку у складі класів та модулів, що повторно використовуються. Вона надає засоби для виділення наскрізної функціональності в окремі модулі, що полегшує роботу із компонентами програмної системи та знижує складність системи в цілому.

Аспектно-орієнтоване програмування (АОП) є однією з концепцій програмування, яка продовжує розвиток процедурного та об`єктно-орієнтованого програмування.

За деякими оцінками, близько 70% часу в проектах витрачається на супровід і внесення змін у готовий програмний код. Тому достатньо важливою у найближчій перспективі стає роль АОП і схожих трансформаційних підходів. Порівняно нова технологія уже отримала досить широке розповсюдження, показавши свою ефективність на тестових аплікаціях, однак місце цього підходу в індустрії ПЗ все ще не визначено. Незважаючи на те, що сьогодні це скоріше академічні дослідження, багато компаній, таких як BEA, IBM, Microsoft, а також open source об'єднання, такі як JBoss, ObjectWeb, Eclipse вже працюють в цьому напрямку з помітними результатами.

При правильному використанні АОП дозволяє [2]: значно зменшити об'єм програмного коду, покращити дизайн та загальну модульність системи, спростити код системи, спростити тестування та супровід, збільшити кількість повторно використовуваних модулів.

АОП – методологія, в основі якої лежать ідеї, що зустрічаються в області технології програмування вже достатньо давно. Це суб'єктно-орієнтоване програмування (subject-oriented programming) [3], композиційні фільтри (composition filters) [4], адаптивне програмування (adaptive programming) [5]. АОП тісно пов'язано з ментальним програмуванням (intentional programming), концепції якого викладені в роботі Чарльза Саймоні [6]. Іншою близькою ідеологією є так зване генеративне та трансформаційне програмування (generative programming, transformational programming) [7].

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010

На даний час досліджено значну кількість різноманітних шляхів виділення функціональностей у складних системах в окремі модулі [1–8]. АОП є одним із цих рішень, він пропонує засоби виділення цих функціональностей в окремі програмні модулі – аспекти.

Аспектно-орієнтований підхід розглядає програмну систему як набір модулів, кожний із яких виражає аспект – ціль, особливість функціонування системи. Набір модулів, що складають програму, залежить від вимог до програми, особливостей її предметної області. При проектуванні програмної системи розробник вибирає модулі так, щоб кожен із них реалізовував певну функціональну вимогу до системи. Але реалізація деяких вимог до програми часто не може бути локалізована в окремому модулі в рамках процедурного чи об'єктно-орієнтованого підходу, в результаті чого код, що відображає такі функціональні аспекти буде знаходитись у різних модулях. Аспектно-орієнтований підхід дозволяє покращити дизайн системи, забезпечуючи можливість локалізації наскрізної функціональності [9] в спеціальних модулях – аспектах. АОП дозволяє реалізувати окремі концепції у слабо зв'язаному вигляді, комбінуючи такі реалізації для формування кінцевої системи. АОП дозволяє побудувати систему, використовуючи слабо зв'язані розбиті на окремі модулі (аспекти) реалізації загальносистемних вимог.

Для розроблення програм за допомогою аспектно-орієнтованого підходу потрібно насамперед відділити функціональність модульного рівня від наскрізної функціональності системного рівня. Наприклад можна реалізувати бізнес-логіку системи за допомогою класів використовуючи об’єктно-орієнтований підхід, а усю наскрізну функціональність використовуючи АОП. Класи об'єктно-орієнтованого підходу (ООП) будуть інкапсулювати у собі основну функціональність системи, а в аспекти буде винесена наскрізна функціональність. Основний код системи і код аспектів буде інтегрований у готовий продукт використовуючи вплітання аспектів (aspect weaving), яке може відбуватися під час компіляції чи динамічно.

На рис. 1 зображено вплітання аспектного і основного коду для отримання виконавчого коду.

Рис. 1. Інтеграція аспектного коду.

Таким чином, за допомогою аспектного вплітання отримують фінальний продукт. Деталі реалізації вплітання залежать від конкретного аспектно-орієнтованого інструменту і засобу.

В попередніх роботах авторами було сконструйовано модуля захисту розподіленої системи теплового проектування з використанням як об'єктно-орієнтованого, так і аспектно-орієнтованого підходів [10–12]

Метою цієї роботи є порівняння АОП та ООП на прикладі реалізації згаданого модуля та аналіз використання АОП як технологічного засобу підвищення надійності програмного забезпечення.

Методика оцінювання та порівняння АОП та ООП

Кількісна оцінка та порівняння реалізацій модуля захисту засобами АОП та ООП здійснювалось на основі метрик програмного коду [1, 13–16]. Вживання метрик дозволяє упорядкувати розробку, випробування, експлуатацію і супровід програмного продукту. Залежно від характеристик і особливостей показника якості застосовуються різні види метрик і шкал для виміру показників.

Перший вид метрик – це метрики, яким відповідає інтервальна шкала, характеризується відносними величинами або реально вимірюваними фізичними показниками. Наприклад, використовуючи цей вигляд метрик можна сказати, що одна програма менш або більш ефективна за іншу на 10 одиниць.

Другий вид метрик – це метрики, яким відповідає порядкова шкала. Вони дозволяють визначати деякі характеристики шляхом порівняння з опорними значеннями, тобто вимір за цією шкалою фактично визначає взаємне положення конкретних модулів або програмних

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010

систем. Для об'єкту вимірювання встановлюється пріоритетність ознак.

Третій вид метрик – це метрики, яким відповідає номінальна шкала. Даний вигляд метрик характеризує наявність певної властивості або ознаки в об'єкті, зокрема в програмному модулі, без врахування градацій за даною ознакою. Такі метрики фіксують наявність певної якості залежно від присутності деякого показника в комплексі програм (наприклад, наявність структуризації, гнучкості, простоти освоєння). Наприклад, таку характеристику як складність модуля можна кількісно оцінити значеннями цієї метрики: [нескладна для розуміння], [помірно складна для розуміння], [складна для розуміння], [дуже складна для розуміння].

Існуючі оцінки програмних систем можна згрупувати за декількома класами:

- Оцінки топологічної і інформаційної складності програми.
- Оцінки рівня мовних засобів і їх вживання.
- Оцінки складності сприйняття і розуміння програмних текстів, орієнтовані на психологічні чинники, істотні для супроводу і модифікації програм.

Для експериментальних досліджень в цій роботі було використано метрики коду, отриманого при програмній реалізації модуля авторизації та автентифікації розподіленої системи теплового проектування з використанням об'єктно-орієнтованого та аспектно-орієнтованого підходів. Значення ряду метрик коду було отримано з використанням автоматизованого програмного засобу Together Architect.

Топологічна та інформаційна складність

Традиційною характеристикою розміру програм є кількість рядків вихідного тексту. Оцінка розміру програм є оцінка за номінальною шкалою, на основі якої визначаються лише категорії програм без уточнення оцінки для кожної категорії. До даної групи оцінок можна віднести метрику Холстеда [13, 14]. Основу цієї метрики складають чотири вимірювані характеристики програми:

- NUOprtr (Number of Unique Operators) – число унікальних операторів програми, включаючи символи-розділювачі, імена процедур і знаки операцій (словник операторів);
- NUOprnd (Number of Unique Operands) – число унікальних операндів програми (словник операндів);
- NOprtr (Number of Operators) – загальне число операторів в програмі;
- NOprnd (Number of Operands) – загальне число операндів в програмі.

Базуючись на цих характеристиках, що отримуються безпосередньо при аналізі вихідних текстів програм, М. Холстед вводить наступні оцінки [13]:

- словник програми (Halstead Program Vocabulary) $\mathrm{HPVoc} = \mathrm{NUOprtr} + \mathrm{NUOprnd}$;
- довжина програми (Halstead Program Length) $\mathrm{HPLen} = \mathrm{NOprtr} + \mathrm{NOprnd}$;
- об'єм програми (Halstead Program Volume) $\mathrm{HPVol} = \mathrm{HPLen} \cdot \log_2(\mathrm{HPVoc})$.

Характеристика складності програми (Halstead Difficulty) обчислюється як $\mathrm{Hdiff} = \mathrm{NUOprtr}/2 \cdot (\mathrm{NOprnd}/\mathrm{NUOprnd})$.

Використовуючи Hdiff Холстед вводить оцінку HEff (Halstead Effort): $\mathrm{HEff} = \mathrm{Hdiff} \cdot \mathrm{HPVol}$, за допомогою, якої описується зусилля програміста при розробці.

Друга найбільш інформативна група оцінок складності програм – метрики складності потоку управління програм. Як правило, за допомогою цих оцінок оперують або щільністю управляючих переходів усередині програм, або взаємозв'язками цих переходів. В обох випадках програма представляється у вигляді графа управління. Вперше графічне представлення програм було запропоноване Маккейбом [14, 15]. Основною метрикою складності він передбачає рахувати цикломатичну складність графа програми, або, як її ще називають, цикломатичне число Маккейба, що характеризує складність тестування програми.

Для обчислення цикломатичного числа Маккейба CC (Cyclomatic Complexity) застосовується вираз:

$$CC = L - N + 2 \cdot P,$$

де L – число дуг орієнтованого графа; N – число вершин; P – число компонентів зв'язності.

До метрик складності також відноситься:

- NORM (Number Of Remote Methods) – кількість віддалених методів, що викликаються. При формуванні значення цієї метрики є видимими всі конструктори і методи класу, і підраховується кількість віддалених методів, що викликаються. Віддаленим методом є метод, який не визначений в класі і його батьках.

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010

- RFC (Response For Class) – відгук на клас – кількість методів, які можуть викликатися екземплярами класу. Ця метрика обчислюється як сума кількості локальних методів і кількості віддалених методів.
- WMPC1 (Weighted Methods Per Class 1) – зважена насиченість класу – дає відносну міру його складності; якщо вважати що всі методи мають однакову складність, то значенням метрики буде кількість методів в класі. Ця метрика визначається сумою складності всіх методів класу, де кожен метод зважується підрахунком його цикломатичного числа. Для розрахунку даної метрики використовуються лише методи визначені в конкретному класі, всі методи, успадковані від батьківського класу, не включаються.
- WMPC2 (Weighted Methods Per Class 2). Ця метрика є мірою складності класу. Вона вважає, що клас з більшою кількістю методів ніж інші є складнішим, і що метод з великою кількістю параметрів також є складнішим. Для розрахунку цієї метрики використовуються лише методи визначені в конкретному класі, всі методи, успадковані від батьківського класу, не включаються.
- LOCOM1 (Lack Of Cohesion Of Methods 1) – нестача зв'язності методів. Зв'язність – це міра взаємодії між елементами окремого модуля, характеристика його насиченості. Високе значення цієї метрики говорить про високу зв'язність методів – це означає, що буде потрібно великі зусилля при тестуванні цих методів, оскільки методи можуть впливати на одні і ті ж атрибути класу. Це також свідчить про низьку готовність до повторного використання.
- LOCOM2 (Lack Of Cohesion Of Methods 2) Зв'язаність методів – міра насиченості абстракції. Клас, який може викликати істотно більше методів, ніж аналогічні за рівнем класи, є складнішим. Метрика підраховує процентне відношення методів, що не мають доступу до специфічних атрибутів класу до всіх атрибутів даного класу. Високе значення зв'язності означає, що клас добре спроектований. Добре зв'язний клас має тенденцію до високої міри інкапсуляції, тоді як відсутність зв'язності зменшує інкапсуляцію і збільшує складність.
- LOCOM3 (Lack Of Cohesion Of Methods 3) Вимірює міру відмінності методів в класі по атрибутах. Низьке значення цього параметру говорить про хорошу декомпозицію в класі, що виражається в його простоті та зрозумілості і здатності до повторного використання. Високе значення говорить про відсутність зв'язності, що збільшує складність, підвищує імовірність помилок в процесі розробки.

У цю групу метрик також потрапляють базові метрики:

- LOC (Lines Of Code) – кількість рядків коду;
- NOC (Number Of Classes) – кількість класів.

Покращення значень метричних характеристик з цієї групи говорить про позитивний ефект від використання методу оптимізації. Наприклад, зменшення значень метрик Холстеда, цикломатичної складності свідчить про те, що отримана реалізація в цілому є кращою з міркувань топологічної складності. Зменшення рівня зв'язності, зваженості методу на клас або відгуку класу говорить про якіснішу функціональну декомпозицію. Зменшення кількості стрічок коду також говорить про позитивний ефект, якщо ту саму функціональність можна викласти за допомогою нового якіснішого підходу з меншою кількістю рядків.

При порівняння аспектної та об'єктної реалізацій за наведеними метриками було отримано результати, які відображені на рис. 2. На цьому рисунку значення метрик ООП реалізації прийнято за 100%, а значення метрик АОП реалізації відображено у відсотках відносно ООП реалізації. Як видно з рис. 2, значення АОП метрик в цілому на 10–40% нижче ніж в ООП реалізації, що позитивним чином впливає на систему, оскільки на кожну метрику (ресурс) буде витрачено менше часу.

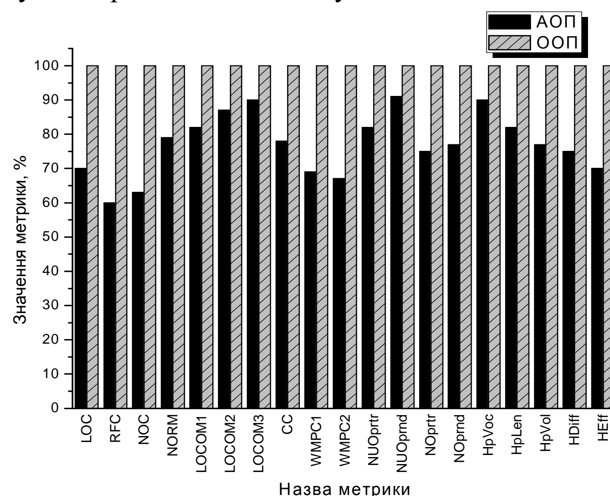

Рис. 2. Зіставлення метрик топологічної складності у АОП і ООП реалізаціях.

З рис. 2 видно, що аспектна реалізація є менша за кількістю стрічок та загальним об'ємом

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010

коду, тоді як кількість унікальних операторів та операндів сильно не відрізняються.

Також у аспектному підході нижча цикломатична складність, що свідчить про те, що аспектна реалізація є краща з точки зору топологічної складності, а також при написанні аспектної реалізації витрачається менше зусиль згідно HEff .

Аспектна реалізація має нижчий рівень зв'язності та складності методів, тому аспектна реалізація містить кращу функціональну декомпозицію.

Рівень мовних засобів та їх вживання, складність сприйняття і розуміння коду

Для оцінки міри застосування аспектного підходу до конкретної програмної системи введемо наступну метрику [16]:

$$\text{Адаптивність} = \frac{Q - P}{Z},$$

де P – розмір коду програмної системи без вживання аспектного підходу; Q – розмір коду компонентів що реалізовують основну логіку системи; Z – розмір коду аспектних модулів, що реалізовують наскрізну функціональність. Отже, в даному випадку значення адаптивності становить:

$$\text{Адаптивність} = \frac{920 - 185}{635} = 1{,}157 .$$

В результаті отримане значення адаптивності становить 1,157, що говорить про позитивний ефект від застосування аспектного підходу до програмної системи.

Метрики складності сприйняття і розуміння програмних текстів не мають кількісної оцінки і характеризують наявність певної властивості або ознаки в об'єкті, зокрема в програмному модулі, без врахування градацій за цією ознакою.

Модульність. Наскрізна функціональність може знаходитися в аспектних модулях, не зачіпаючи при цьому модулі, що реалізовують основні функціональні вимоги, тобто всі залежності між кодом реалізованому в аспекті і компонентами локалізовані в коді аспекту. Компоненти, що беруть участь, абсолютно вільні від контексту і як наслідок ці компоненти повністю готові до повторного використання. У [1] модульність описується як "властивість системи, яка була розкладена на внутрішньо зв'язні, але слабо зв'язані між собою модулі". Правильне розділення програми на модулі є таким же складним завданням, як вибір правильного набору абстракцій.

Здатність до повторного використання. Компоненти, які не зв'язані між собою специфічною для даної програмної системи наскрізною функціональністю, легко можуть бути використані повторно. Причому в різних частинах системи один компонент може грати різні ролі, залежно від аспектних модулів. Крім того, код аспектів і їх складових частин також можна повторно використовувати за допомогою наслідування і це дає ще ширші можливості.

Зрозумілість коду. Оскільки наскрізна функціональність винесена в аспектні модулі, композиція класів в системі виглядає зрозумілішою. Це пов'язано з тим, що реалізація класів необтяжена кодом, який не відноситься безпосередньо до сутності класу.

У таблиці наведено порівняння реалізацій за даним набором метрик.

Таблиця

Порівняння реалізації за складністю сприйняття

Метрики	АОП	ООП
модульність	+	+/–
повторного використання	+	+/–
зрозумілість коду	+/–	+

Аналізуючи цей набір метрик можна зробити висновок, що код написаний з використанням АОП має кращу модульність та здатність до повторного використання, але є гіршим з точки зору розуміння коду.

Провівши порівняння АОП і ООП реалізацій по усіх наведених вище метриках можна зазначити, що АОП надає такі переваги:

- менший об'єм коду;
- краща модульність;
- хороша адаптивність коду;
- нижчий рівень складності коду;
- краща функціональна декомпозиція коду;

Недоліком стало те, що код написаний використовуючи АОП є складнішим для розуміння, важче визначити потік виконання методів.

Висновки

В роботі проведено порівняння ефективності застосування аспектно-орієнтованого підходу з об'єктно-орієнтованим за рядом метрик. Порівняння за метриками топологічної складності показало, що при застосуванні АОП вони на 10–40% нижчі, ніж при застосуванні

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010

ООП, тобто код написаний з використанням АОП має нижчу топологічну складність. На основі проведеного порівняння було виділено, що застосування АОП надало такі переваги:

- менший об'єм коду;
- краща модульність;
- хороша адаптивність коду;
- нижчий рівень складності коду;
- краща функціональна декомпозиція.

Недоліком стало те, що код написаний використовуючи АОП є складнішим для розуміння, важче визначити, потік виконання методів.

Таким чином застосування АОП дозволяє покращити якість, надійність та розширюваність програмних систем. Крім того,нижча топологічна складність, менший об'єм коду, нижчий рівень складності коду та краща функціональна декомпозиція дозволяють зробити обґрунтоване припущення, що використання АОП покращує надійність програмного забезпечення за рахунок меншої складності проекту [17].

Список літератури

1. *Г. Буч*. Объектно-ориентированный анализ и проектирование. – СПб.: Издательство Бином, Невский диалект, 1998. – 560 с.

2. *G. Kiczales, J. Lamping, A. Mendhekar*, et al. Aspect-oriented programming // Proceedings of the ECOOP'97 Conference, LNCS 1241, Springer-Verlag, 1997.

3. Homepage of the Subject-Oriented Programming Project, [Електронний ресурс]. – Режим доступу: http://www.research.ibm.com/sop/.

4. *M. Aksit, L. Bergmans*, and *S. Vural*. An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach // Proceedings of the ECOOP'92 Conference, LNCS 615, Springer-Verlag, 1992.

5. *K. Leiberherr*. Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes // Information Processing '92, 12th World Computer Congress, Madrid, Spain, J. van Leeuwen (Ed.), Elsevier, 1992, pp.179–185.

6. *Ch. Simonyi*. The Death of Computer Languages, The Birth of Intentional Programming, [Електронний ресурс]. – Режим доступу: research.microsoft.com/apps/pubs/default.aspx?id=69540.

7. *Krzysztof Czarnecki, Ulrich Eisenecker*. Generative Programming: Methods, Tools, and Applications. – Addison-Wesley, Paperback, Published June 2000.

8. *E. Dijkstra*. Programming Considered as a Human Activity. Classics in Software Engineering. – New York, Yourdon Press, 1979.

9. Aspect-Oriented software development network. [Електронний ресурс]. – Режим доступу : http://www.aosd.net.

10. *Одуха О.В., Яковина В.С.* Проектування системи захисту розподіленої системи теплового проектування // Матеріали Третьої Міжнародної конференції "Комп'ютерні науки та інформаційні технології" CSIT'2008, Львів, 2008, с. 339–341.

11. *Яковина В.С., Мамроха Н.М., Сенів М.М.* Аспектна декомпозиція компонентів захисту розподіленої системи теплового проектування // Збірник матеріалів шостої міжнародної конференції "Інтернет – Освіта – Наука – 2008" ІОН-2008, Вінниця, Том 2, с. 407–410.

12. *Федасюк Д.В., Яковина В.С., Сенів М.М., Мамроха Н.М.* Побудова моделі аспекту аутентифікації та авторизації для підсистеми захисту програмних систем // Матеріали 4-ї Міжнародної науково-технічної конференції "Комп'ютерні науки та інформаційні технології" CSIT-2009, Львів, 2009, с. 198–202.

13. *Холстед М.* Начала науки о программах. – М.: Финансы и статистика, 1981. – 128 с.

14. *Изосимов А.В., Рыжко А.Л.* Метрическая оценка качества программ. – М.: МАИ, 1989. – 96 с.

15. *Новичков А*. Метрики кода и их практическая реализация в IBM Rational ClearCase [Електронний ресурс]. – Режим доступу: http://www.viva64.com/go.php?url=241

16. *Павлов В*. Анализ вариантов применения аспектно-ориентированного подхода при разработке программных систем, [Електронний ресурс]. – Режим доступу: http://www.javable.com/columns/aop/workshop/01/

17. *Я.М. Чабанюк, В.С. Яковина, Д.В. Федасюк, М.М. Сенів, У.Т. Хімка* Побудова і дослідження моделі надійності програмного забезпечення з індексом величини проекту // Інженерія програмного забезпечення. – № 1 (2010). – С. 24-29.

Відомості про авторів:

Яковина Віталій Степанович, доцент, кафедра програмного забезпечення, Національний університет "Львівська політехніка"; кандидат фізико-математичних наук; наукові інтереси – надійність та безпека програмного забезпечення
E-mail: yakovyna@polynet.lviv.ua

Федасюк Дмитро Васильович, проректор, Національний університет "Львівська політехніка", кафедра програмного забезпечення; доктор технічних наук; наукові напрями – автоматизація теплового проектування мікроелектронних систем, технології створення програмного забезпечення
E-mail: fedasyuk@lp.edu.ua

Мамроха Назарій Михайлович, студент ОКР "магістр", кафедра програмного забезпечення, Національний університет "Львівська політехніка", наукові інтереси – аспектно-орієнтоване програмування
E-mail: nazhappy2003@yandex.ru

Стаття надійшла до редакції 18.10.2010 року

© В.С. Яковина, Д.В. Федасюк, Н.М. Мамроха, 2010