УДК 004.415.2.045 (076.5)

ОЦІНЮВАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ПРОГРАМНИХ СИСТЕМ НА ЕТАПІ ПРОЕКТУВАННЯ

Радішевський М.Ф., Рябокінь Ю.М.
Національний авіаційний університет
mykola.Radshevskiy@live.nau.net

У статті розглянуто об'єктно-орієнтовані метрики, які використовуються для оцінювання складності програмних систем на етапі проектування. Досліджено застосування метрик (NSS, NKC, NSUB, DIT, NOC, CBO, RFC, WMC) та їх вплив на якісні характеристики програмного забезпечення. Програмні системи володіють наслідуваною складністю, яка має бути оцінена і зменшена на ранніх етапах життєвого циклу. Для полегшення цього завдання широко використовуються відповідні метрики програмного забезпечення.

This paper reviews object-oriented metrics for software design assessment from the complexity point of view. Applied metrics (NSS, NKC, NSUB, DIT, NOC, CBO, RFC, WMC) is studied and their influence on software quality characteristics is investigated. Software systems possess inherited complexity which should be estimated and reduced early in their life cycle. Software metrics are widely used to facilitate this tas.

Вступ

У своїй практичній діяльності людина постійно взаємодіє з різними штучними та реальними системами. Ця взаємодія може бути простим спостереженням за системою, вивченням (аналізом) існуючої або створенням (синтезом) нової системи. Під системою розуміють сукупність зв'язаних і взаємодіючих елементів будь-якої природи. Прикладами реальних систем є атом, організм. Прикладами штучних систем є комп'ютер, програма.

Постановка завдання

Одна з проблем аналізу систем — оцінювання їх складності. Складність системи є якісною характеристикою. Зменшення складності ПС дає змогу знизити трудомісткість проектування, розробки, тестування та супроводження, забезпечує простоту і надійність виробленої ПС. У даний час питанням управління якістю програмних систем (ПС) приділяється підвищена увага, пов'язано це з ростом застосувань програмних систем у різних сферах діяльності людини. Управляння якістю включає в себе планування якості, забезпечення якості і контроль якості. Таким чином, однією зі складових розробки ПС із запланованим рівнем якості є контроль, який базується на застосуванні метрик якості для вимірювання основних показників в процесі розробки ПС.

Метрика програмного забезпечення — міра, яка дає змогу отримати числове значення деякої властивості програмного забезпечення або його специфікацій [1].

Специфічним видом метрик якості ПС є метрики складності. Особливість даних метрик полягає в тому, що вони прямо чи непрямо впливають на ряд інших метрик якості і на більшість характеристик якості ПС. У першу чергу на надійність, супроводжуваність та ефективність. Одним із шляхів контролю і досягнення необхідного рівня якості ПС є оцінювання складності проміжних продуктів у процесі розроблення ПС, яка допомагає в подальшому керувати процесом.

На проектування, кодування та тестування припадає більш ніж 75 % вартості конструювання ПС [2]. Прийняті на цьому етапі рішення здійснюють вирішальний вплив на успіх реалізації ПС та легкість, з якою ПС супроводжуватиметься. Важливість проектування можна визначити словом якість, тобто проектування — це етап, на якому «вирощується» якість ПС.

Проектування забезпечує правильну трансляцію вимог замовника в кінцевий програмний продукт. Тому для досягнення відповідного рівня якості ПС доцільно здійснювати оцінку складності ПС на етапі проектування ПС для зменшення кількості помилок та часу на їх ліквідацію.

Життєвий цикл розроблення ПС із включеним процесом оцінювання складності ПС зображено на рис. 1.

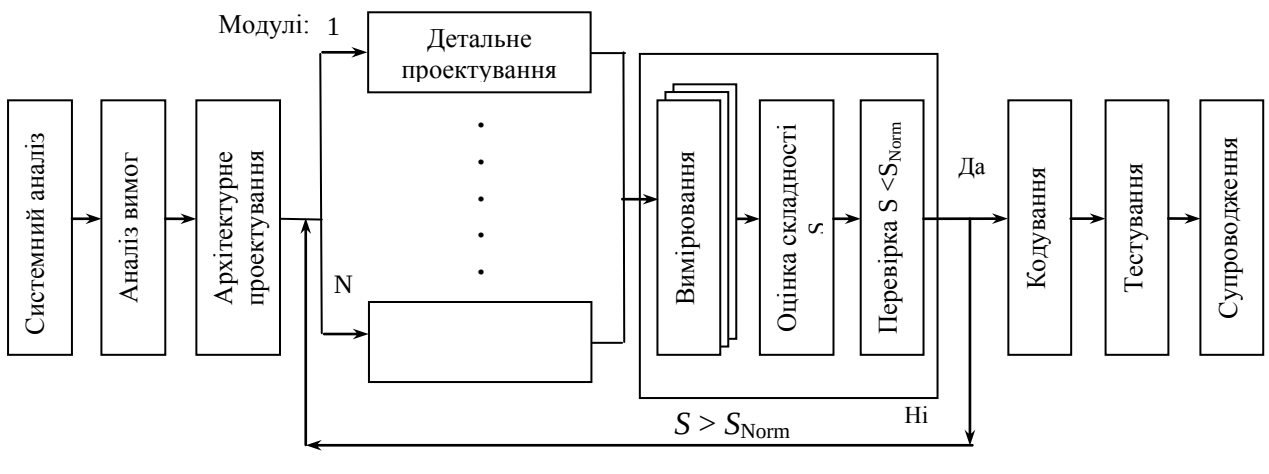

Рис. 1. Структурна схема розробки ПС

Проектування — ітераційний процес, за допомогою якого вимоги до ПС транслюються в інженерні представлення ПС. На початку ці представлення дають тільки концептуальну інформацію (на високому рівні абстракції), наступні уточнення приводять до форм, близьких до текстів на мовах програмування.

У проектуванні виділяють два рівну проектування: *архітектурне* та *детальне проектування*. Архітектурне проектування формує абстракції архітектурного рівня, детальне проектування уточнює ці абстракції (рис. 1).

Архітектурне проектування включає три етапи діяльності:

1) структурування системи — розподіл системи на кілька підсистем (незалежний програмний компонент) та визначення взаємодії підсистем;

2) моделювання управління — визначення моделі зв'язків між частинами системи;

3) декомпозиція підсистем на модулі — поділ підсистем на модулі (визначення типів модулів та міжмодульних з'єднань).

Розгляд будь-якої складної системи вимагає застосування техніки декомпозиції — поділ на складові елементи.

Існують такі схеми декомпозиції: алгоритмічна та об'єктно-орієнтована. В основі алгоритмічної декомпозиції лежить розподіл по діям — алгоритмам.

Така схема застосовується в звичайних ПС.

Об'єктно-орієнтована декомпозиція забезпечує поділ за автономними особами — об'єктами реального світу. На етапі об'єктно-орієнтованої декомпозиції визначаються класи об'єктів, їх властивості та операції. При реалізації системи із цих класів створюються об'єкти; для координації операцій об'єктів використовується будь-яка модель управління (модель централізованого управління, модель управління повідомленнями).

До переваг об'єктно-орієнтованого підходу відносять [2]:

- слабкий зв'язок між об'єктами (можна змінювати реалізацію того чи іншого об'єкта не впливаючи на інші об'єкти);
- легкозрозуміла структура системи, оскільки об'єкти часто є об'єктами реального світу;
- для безпосередньої реалізації системних компонентів можна використовувати об'єктно-орієнтовані мови програмування.

Процес об'єктно-орієнтованого проектування складається з етапів проектування архітектури, ідентифікації об'єктів системи, опису архітектури різноманітними моделями об'єктів і документування інтерфейсу об'єктів [3].

У процесі об'єктно-орієнтованого проектування можна створювати різні *статичні* та *динамічні* моделі.

Статичні моделі описують статичну структуру системи в термінах класів об'єктів і взаємовідношень між ними (моделі класів, моделі узагальнення, моделі агрегації); динамічні моделі описують динамічну структуру системи і показують взаємодію між об'єктами системи (моделі послідовностей, моделі кінцевого автомату).

Оскільки основними елементи конструювання об'єктно-орієнтованих програм є класи, які реалізують свою функціональність методами, то саме класи та методи є категоріями, якими переважно оперують об'єктно-орієнтовані метрики.

Оцінювання складності ПС на етапі проектування здійснюється як оцінювання складності її окремих компонентів (класів) та зв'язків між ними. Аналіз існуючих метрик показує, що для оцінювання складності ПС на етапі проектування застосовують такі об'єктно-орієнтовані метрики [4]:

1) метрики зв'язності класів (TCC, LCC);

2) метрики зчеплення класів (CBO);

3) комплексні набори метрик складності та якості ПС (WMC, RFC, DIT, NOC, NC).

Зв'язність та зчеплення класів мають аналогію зі зв'язністю та зчепленням модулів. Класи відповідають модулям, а функції — методам. Збільшення внутрішньої зв'язності та зменшення зовнішнього зчеплення знижує складність та сприяє забезпеченню надійності програмних систем.

Д. Біємен та Б. Кенг запропонували метрики зв'язності класу, які базуються на прямих та непрямих з'єднаннях між парами методів [5]. Якщо існують загальні екземплярні змінні, що використовуються в парі методів, то вважають, що ці методи з'єднані прямо. Пара методів може бути з'єднана непрямо, через інші прямо зв'язані методи.

На рис. 1 показано відношення між елементами класу *Stack*. Прямокутниками позначені методи класу, а овалами — екземплярні змінні. Зв'язки пов'язують відношення між методами та змінними.

Зі схеми (рис. 2) видно, що методи *Push, Pop, Vtop* і *IsEmpty* попарно з'єднані прямо. А методи *Size* та *Pop* з'єднані непрямо. Для формалізації моделі Д. Бієменем і Б. Кенгом були введені поняття абстрактного методу та абстрактного класу. Абстрактний метод (АМ) — представлення реального методу М у вигляді екземплярних змінних, які прямо чи непрямо використовуються методом. Абстрактний клас (АС) — це представлення реального класу С у вигляді сукупності абстрактних

методів, причому кожний абстрактний метод відповідає видимому методу класу С.

Метрики зв'язаності класу подаються в такому вигляді:

1) TCC (Tight Class Cohesion) — сильна зв'язність класу, визначається відносною кількістю прямо з'єднаних методів:

$$TCC(C) = NDC(C) / NP(C),$$

де NDC(C) — кількість прямих з'єднань AC(C), NP(C) — загальна кількість пар абстрактних методів в AC(C).

Загальна кількість пар абстрактних методів NP(C) для класу С обчислюється як:

$$NP(C) = N(N - l)/2,$$

де N — кількість методів.

2) LCC (Loose Class Cohesion) — слабка зв'язність класу, визначається відносною кількістю прямо чи непрямо з'єднаних методів:

$$LCC(C) = (NDC(C) + NIC(C)) / NP(C),$$

де NIC(C) — кількість непрямих з'єднань в AC(C).

Застосуємо до класу *Stack* (рис. 2) метрики зв'язності, отримаємо такі значення:

TCC (Stack) = 7/10 = 0,7 LCC(Stack) = 10/10 = l.

Метрика TCC показує, що 70% видимих методів класу Stack з'єднані прямо, а метрика LCC показує, що всі видимі методи класу Stack з'єднані прямо або непрямо.

CBO (*Coupling Between Object classes*-CBO) — зчеплення між класами об'єктів, дає змогу визначити кількість класів, з якими зв'язаний даний клас. Це означає, що один клас використовує методи та екземпляри другого класу.

На схемі (рис. 3) зображено структуру класу, яка використовується для наочного визначення зчеплення між класами об'єктів CBO [2].

Зі схеми (рис. 3) видно, що CBO(A) дорівнює 1, оскільки як він використовує один метод із другого класу (метод op_e() із класу Class E, він викликається із методу op_a3()).

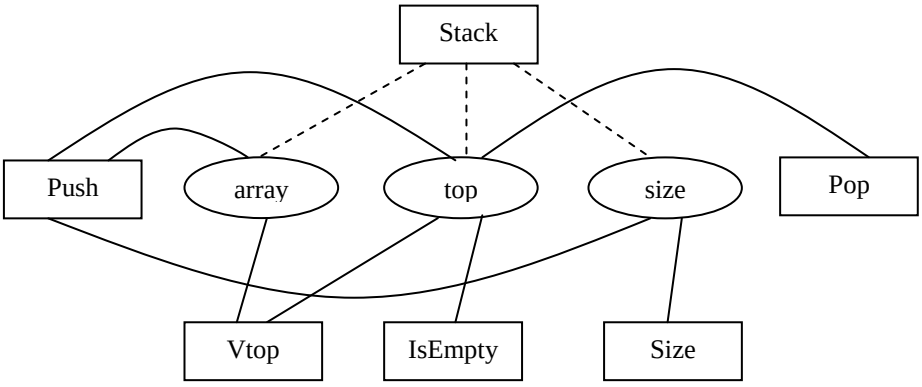

Рис. 2. Відношення між елементами класу *Stack*

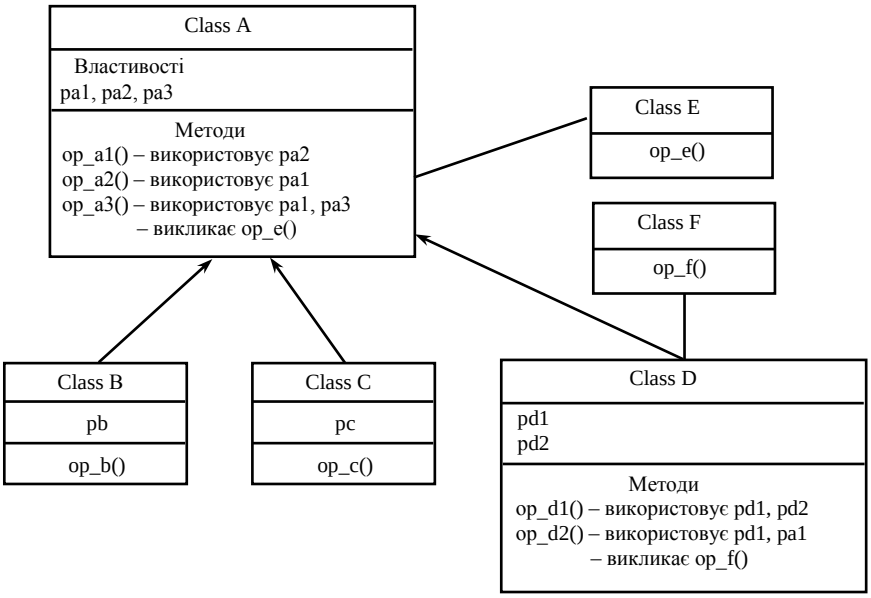

Рис. 3. Структура класів

WMC (*Weighted Methods Per Class*) — кількість «вагових» методів на клас, дозволяє вирахувати складність класів з урахуванням склад-ності його методів. Вагою метода в такому випадку називається кількісна характеристика (метрика) складності методу.

Для оцінювання складності метода може бути обрана будь-яка метрика складності, зокрема — цикломатична складність.

Відповідно WMC визначається за формулою:

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

де C_i — складність *i*-го методу.

Кількість методів і їх складність є індикатором затрат на реалізацію і тестування класів [2]. Крім того, чим більше методів, тим складніше дерево наслідування (усі підкласи наслідують методи базового класу). Із ростом кількості методів у класі його застосування стає більш специфічним, тим самим обмежується можливість багаторазового використання. Тому метрика WMC повинна мати розумно низьке значення.

Існує два варіанта підрахунку кількості методів у класі:

1) підраховуються методи тільки поточного класу. Успадковані методи ігноруються.

2) підраховуються методи, визначені в поточному класі, і всі успадковані методи.

Дуже часто використовують різновидність метрики WMC, коли всі методи мають однакову вагу, при цьому вважають, що $C_i = 1$, вона називається метрикою NM (*Number of Methods*) — кількість методів на клас. Використовується для вимірювання складності класів на ранніх етапах розробки системи, коли іще відсутня детальна інформація про методи.

Загалом клас, що має максимальну кількість методів серед класів одного з ним рівня ієрархії, є найскладнішим.

WMC для класу (*Class* A), зображеного на рис. 3 дорівнює 3. Так як *Class* A має три методи, зі складністю, що дорівнює 1.

RFC (*Responce For Class*) — кількість реакцій на клас допомагає визначити кількість методів, які можуть бути виконані у відповідь на отримані повідомлення даним класом. У цій метриці враховуються не тільки методи даного класу, що виконуються, а й методи інших класів. Таким чином за допомогою цієї метрики можна також визначити ступінь потенційного «комунікування» цього класу з іншими класами. Метрика RFC для *Class* A, зображеного на рис. 3, дорівнює 4, бо у відповідь на прибуття в цей клас повідомлень можливе виконання чотирьох методів. DIT (*Depth of Inheritance Tree* — DIT) — глибина дерева наслідування. Допомагає визначити кількість класів-нащадків, які потенційно впливають на даний клас.

DIT визначається як максимальна довжина шляху від листка до кореня дерева наслідування класів. Наприклад, для ієрархії класів, що зображена на рис. 4 метрика DIT дорівнює 3.

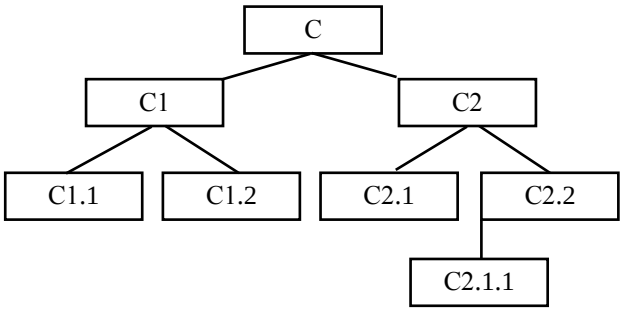

Рис. 4. Дерево наслідування класу

NOC (*Number Of Child* — NOC) — кількість нащадків, допомагає визначити кількість безпосередніх нащадків даного класу.

Підкласи, які безпосередньо підпорядковуються суперкласу (базовому класу), називаються його нащадками (похідними класу). Значення NOC дорівнює кількості нащадків, тобто кількості безпосередніх похідних класів в ієрархії класів.

Для рис. 3 NOC (C2) дорівнює 2. Тобто клас C2 має двох нащадків — підкласи C2.1 і C2.2.

Метрики DIT і NOC — кількісні характеристики форми та розміру ієрархії класів. Добре структурована об'єктно-орієнтована система найчастіше буває організована не як високе дерево, а як сукупність елементів в один ряд (як ліс).

Основні об'єктно-орієнтовані метрики, зазначені вище, застосовуються для оцінювання складності відповідних артефактів етапу проектування. Метрики для оцінювання складності ПС та аналіз їх впливу на характеристики системи зведено в табл. 1.

Наряду з об'єктно-орієнтованими метриками оцінювання складності ПС використовують метрики процесу розробки (проектування, контролю, координації):

- NSS (*Number of Scenario Scripts*) — метрика визначення кількості описаних сценаріїв;
- NKC (*Number of Key Classes*) — метрика кількості ключових класів;
- NSUB (*Number of Subsystem*) — метрика кількості підсистем.

NSS (*Number of Scenario Scripts*) — кількість описаних сценаріїв. Ця кількість прямо пропорційна кількості класів, необхідних для реалізації вимог, кількості станів для кожного класу, а також кількості методів, властивостей і співробітництва. Метрика NSS — ефективний індикатор розміру програм. Рекомендоване значення NSS — не менш одного сценарію на протокол підсистеми, що відображає основні функціональні вимоги до підсистеми.

NKC (*Number of Key Classes*) — кількість ключових класів. Ключовий клас прямо пов'язаний з комерційною проблемною областю, для якої призначена система.

Значення NKC достовірно відображає майбутній обсяг розробки. Кількість ключових класів визначається безпосереднім підрахунком. Рекомендована кількість ключових класів в проекті має складати не менше 20 % від

загальної кількості класів. У протилежному випадку вважають, що виділені класи необхідно передивитися (поглибити дослідження проблемної області).

NSUB (*Number of SubSystem*) — кількість підсистем. Кількість підсистем визначається безпосереднім підрахунком, та забезпечує розуміння питань розміщення ресурсів, планування, загальні затрати на інтеграцію.

Рекомендується виділяти в програмній системі не менше трьох підсистем (NSUB > 3).

Таблиця 1

Метрики для оцінювання складності ПС

Метрика	Способи отримання	Вплив на характеристики системи
CBO (зчеплення між класами об'єктів)	вимірюванням	— занадто велика зв'язність класів впливає на модульність проекту і не дозволяє повторно використати класи; — велика кількість взаємозв'язків збільшує залежність інших частин системи від даного класу та ускладнює супроводження системи в цілому; — сильно взаємозв'язана система вимагає великої кількості тестів та часу на тестування
WMC (кількість «вагових» методів на клас)	обчисленням	— за допомогою метрики WMC оцінюється складність класу. Тобто оцінюється час і зусилля, які необхідно витратити на його розроблення та супроводження, враховуючи складність його методів; — велика кількість методів базового класу потенційно розповсюджує свій вплив і на нащадків цього класу. Оскільки всі нащадки наслідують сі методи базового класу, то ускладнення методів базового класу може відбитися на нащадках классу; — класи з великою кількістю методів з великою часткою ймовірності специфічні для додатка. Тому скоріше за все вони будуть менш придатні для повторного використання в інших додатках
RFC (кількість реакцій на клас)	обчисленням	— велика кількість методів, які викликаються при виклику деякого методу класу, суттєво ускладнює тестування і налагодження цього классу; — показник з найгіршим значенням цієї метрики вимагає найбільшого часу тестування. З ростом RFC збільшується складність класу
DIT (глибина дерева наслідування)	вимірюванням	— велика кількість нащадків робить поведінку класу менш передбачуваною; — глибокі дерева наслідування ускладнюють проект, оскільки включають велике кількість атрибутів і методів у классах-нащадках; — чим глибше положення класу в дереві ієрархії, тим більше повторне використання його методів
NOC (кількість нащадків)	вимірюванням	— чим більше нащадків класу, тим більше повторне використання його методів; — чим більше нащадків класу, тим більша ймовірність неправильного використання базового классу; — чим більше нащадків класу, тим більший вплив він здійснює на систему в цілому. Такий клас вимагає ретельнішого тестування

Висновки. У ході проведених досліджень проаналізовано сукупність об'єктно-орієнтованих метрик на етапі детального проектування, проаналізовано їх вплив на характеристики програмних систем. Серед існуючих об'єктно-орієнтованих метрик, були виділені такі, які використовуються для оцінювання складності на етапі проектування ПС (NSS, NKC, NSUB, DIT, NOC, CBO, RFC, WMC). Існуючі метрики дають можливість оцінити окремі характеристики складності ПС, але для оцінювання складності ПС загалом необхідне використання зазначених метрик у комплексі, для прийняття об'єктивного рішення на етапі проектування. Це дасть змогу вчасно внести відповідні корективи до проекту, що, можливо, значно спростить та прискорить роботу.

ЛІТЕРАТУРА

1. http://ru.wikipedia.org/

2. *Орлов С.А.* Технологии разработки программного обеспечения : учеб. / С.А. Орлов. — СПб. : Питер, 2002. — 464 с.

3. *Соммервилл И.* Инженерия программного обеспечения. — 6-е изд.: пер. с англ. / И. Соммервилл. — М.: Изд. дом «Вильямс», 2002. — 624 с.

4. *Kan S.H.* Metrics and Models in Software Quality Engineering, Second Edition.: Addison Wesley, 2002. — 560 p.

5. *Biem J.M., Kang B-K.* Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symposium on Software Reusability(SSR'95), 1995. — P. 259—262.

Стаття надійшла до редакції 20.03.09