https://doi.org/10.31891/2219-9365-2023-75-11
УДК 004.051

ФОРКУН Юрій
Хмельницький національний університет
https://orcid.org/0000-0002-7906-4191
forkun@ridne.net
МАРТИНЮК Валерій
Хмельницький національний університет
https://orcid.org/0000-0001-5758-4244
martynyuk.valeriy@gmail.com
ПРАВОРСЬКА Наталія
Хмельницький національний університет
https://orcid.org/0000-0001-6001-3311
margana2000007@gmail.com
ЛУЧИЦЬКИЙ Олег
Хмельницький національний університет
oleg.arch999@gmail.com

МЕТРИКА ДИФЕРЕНЦІЙОВАНОЇ ЦИКЛОМАТИЧНОЇ СКЛАДНОСТІ АНАЛІЗУ ПРОГРАМНГО КОДУ З ВИКОРИСТАННЯМ СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ

У даній роботі розглянуті сучасні рішення в області проведення аналізу програмного коду, виділені їх переваги та недоліки. Досліджені існуючі метрики складності коду на предмет їх комбінування і досягнення кращих результатів, у результаті чого була запропонована метрика диференційованої цикломатичної складності, основана на комбінації метрик цикломатичної складності та метрики Чепіна.

Ключові слова: аналіз програмного коду, цикломатична складність, метрика Чепіна, база даних, інженерія програмного забезпечення, якість програмного забезпечення.

FORKUN Yuriy, MARTYNYUK Valeriy, PRAVORSKA Nataliia, LUCHYTSKYI Oleh
Khmelnytskyi National University

METRICS OF DIFFERENTIATED CYCLOMATIC COMPLEXITY OF SOFTWARE CODE ANALYSIS USING DATABASE MANAGEMENT SYSTEMS

In the modern world, the level of software development has reached a sufficiently high standard, making the question of quality software development particularly relevant. Specifically, the issue of analyzing the code of a software application allows for a thorough examination of every line of source code to understand its characteristics, quality, and potential issues. The quality and reliability of a program, as well as its future success, depend on the analysis of the source code.

This is especially true for today's popular software with a multi-tiered structure, where various algorithmic elements, such as databases, are distributed among different applications, requiring special attention to data exchange implementation.

This article explores modern solutions in the field of code analysis, highlighting their advantages and disadvantages. Existing code complexity metrics are examined for possible combinations to achieve better results, resulting in the proposal of a metric called "Differential Cyclomatic Complexity," which is based on the combination of cyclomatic complexity and Chapin's metric.

Keywords: software code analysis, cyclomatic complexity, Chapin's metric, database, software engineering, software quality.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Поняття інженерії програмного забезпечення (англ. «Software engineering») існує вже більше півсторіччя. Діяльність в межах цієї дисципліни формує основу, практично, для всієї науки розробки програмного забезпечення, включаючи в себе весь життєвий цикл створення програмних додатків, починаючи від аналізу вимог та проектування і закінчуючи тестуванням, підтримкою та оновленням. Розповсюдження програмного забезпечення в наш час у майже у всі сфери людського життя – результат досконалого вивчення та використання знань, досягнутих в межах цієї дисципліни.

Одним з питань, які досліджує інженерія програмного забезпечення, є питання якості розроблюваного програмного забезпечення. Одним з ведучих чинників є якість програмного коду. Запитання, що виникають перед розробниками програмного забезпечення, завжди були важливими і складними. Як зробити код більш надійним? Як забезпечити його продуктивність та ефективність? Як підтримувати та розвивати застосунок з плином часу? Як переконатися, що вона працює коректно та безпечно?

Аналіз вихідного коду відіграє ключову роль у відповіді на ці запитання. Це процес, який дозволяє ретельно розглянути кожен рядок програмного коду та зрозуміти його характеристики, якість та можливі проблеми. Від аналізу вихідного коду залежить якість та надійність програми, а також її подальший успіх.

Сучасні програмні застосунки стають все складнішими, а підтримка розроблюваного програмного забезпечення сьогодні супроводжуються створенням оновлень з відносно великою частотою. Не дивлячись на роботу, напрямлену, здебільшого, на покращення роботи вже створеного функціоналу – видалення помилок та додавання нових можливостей – регулярно виникає ситуація, коли чергове оновлення призводить до погіршення роботи алгоритму, яке доводиться неминуче виправляти. У таких обставинах аналіз вихідного коду стає настільки ж важливим елементом процесу розробки, наскільки й складним. Зростання обсягу коду, використання різних технологій та бібліотек, а також вимог до продуктивності та безпеки роблять аналіз вихідного коду необхідністю.

Окремої уваги заслуговує програмного забезпечення, в основі роботи якого лежить принцип розділення алгоритмів роботи між безпосереднім додатком та базою даних. Ця структура ще називається моделлю «клієнт-сервер». В такій архітектурі за реалізацію різних елементів програмного застосунку відповідають різні додатки, між якими відбувається обмін інформацією. В таких застосунках, окрім роботи основного коду, аналізу потребують алгоритми обміну даними між програмними застосунками які працюють з базами даних, системами керування базами даних та програмних застосунків з багаторівневою структурою.

В даній статті будуть розглянуті сучасні пропозиції щодо вирішення питання аналізу програмного коду, визначені їх переваги та недоліки, на основі яких буде запропонована ще одна методика проведення аналізу коду програмних застосунків, яка продемонстрована у вигляді метрики диференційованої цикломатичної складності, що створена на основі показників метрики цикломатичної складності та метрики Чапіна.

Аналіз досліджень та публікацій

Аналіз програмного коду є багатофакторною областю у розробці програмних застосунків, тобто існують різні варіанти впровадження нових рішень у цю область. Це спостерігається через ознайомлення з різноманітними роботами за даною тематикою, куди входить огляд таких тем:

– безпека програмного забезпечення;
– розробка засобів аналізу коду;
– розробка нових метрик;
– профілювання програмного забезпечення та інші.

Розглянемо частину цих робіт. Почнемо з роботи [1]. У даній статті автори розглядають явище регресії продуктивності роботи програмного застосунку, яка може виникати у випадку значних змін у коді в процесі створення нової версії. Вирішення даної проблеми представлені у вигляді використання або специфічних юніт-тестів, які самі по собі є непопулярним рішенням, або методик, які не користуються кодом та мають високі вимоги для отримання можливості проведення аналізу. Тому авторами була запропоновано прототип власного інструменту, робота якого базується на аналізі коду. Таке впровадження, згідно тексту роботи, зменшує витрати часу у процесі тестування та складність його проведення.

У роботі [2] досліджується аналіз коду на предмет наявності шкідливого коду, який є наслідком поширення тенденції використання відкритого вихідного коду у процесі розробки, що створює вразливості у системі безпеки програмного забезпечення. Для вирішення цієї проблеми, автори пропонують алгоритм, який використовує нейронну мережу з глибоким навчанням для аналізу коду для пошуку аномалій, які вказуватимуть на наявність вразливостей у програмному застосунку. Хоча автори зазначають, що метод має недоліки, але їх робота впроваджує сучасний засіб захисту, пов’язаний з актуальною проблемою.

У ще одній роботі [3] автори також досліджують пошук вразливостей у коді, мотивуючи це сучасними тенденціями (вузькі часові межі та слабо контрольована робота розробника з дому), що погіршують ефективність традиційних методик тестування програмного забезпечення. В цій роботі, технологію штучного інтелекту було об’єднано з технологією блокчейнів, яка дозволить підтримувати взаємодію з розроблюваним програмним забезпеченням та її складовими в умовах дистанційною роботи, підвищуючи рівень безпеки при обміні інформацією через мережу. Разом з автоматизацією процесу аналізу коду штучним інтелекту, нова методика полегшує роботу поза робочими приміщенням компаній-розробників.

Робота [4] пов’язана з добуванням інформації з вихідного коду об’єктно-орієнтованих мов програмування. Автори роботи розглядають існуючі методики для досягнення цих цілей, звертаючи увагу на обмеженість цих засобів в таких моментах, як доступність цих самих засобів для потенційних користувачів, так й обсягу та якості інформації, яка надається в результаті використання цих засобів. Впроваджений авторами новий метод базується на зовсім інших, ніж у традиційних попередників, технологіях, завдяки чому досягаються кращі результати у процесі збору вихідного коду та вилученні

необхідної розробникам інформації. Враховуючи перехід на нові принципи створення моделі, це робота може бути основою для якісного стрибка серед засобів аналізу коду.

Автори роботи [5], розглядаючи оцінку складності, читабельність та зрозумілості коду, пропонують нову метрику у чотирьох варіаціях, яка дозволяє аналізувати код підвищеної складності, зокрема – код, в якому активно використовуються рекурсії, до яких автори звертають особливу увагу. Очевидно, що використання нової метрики робить зручнішим аналіз розвиненого коду, з яким можуть виникнути складності при використанні більш старіших метрик.

В тексті роботи [6] розглядає аналіз коду з точки зору т. зв. підсумування вихідного коду (англ. «Source Code Summarization»). Звертаючи увагу на можливість покращення якості підтримки програмного забезпечення та його розвитку за допомогою якісного коментування та опису коду, автори вказують на загальну низьку ефективність описування коду традиційними методами: API документація, GitHub та опис вручну. Таким чином, пропонується використання методу на основі принципу доповнення даних (англ. «Data Augmentation»). Згідно тексту роботи, новий метод дозволяє генерувати високоякісний опис програмного коду, що надає перспективи полегшити роботу при підтримці коду.

Огляд останніх досліджень та робіт в області аналізу коду вказує на логічні тенденції: автори робіт використовують досягнення останніх часів (в основному – поширення нейронних мереж) або розробляють власні методики, базуючись на недоліках існуючих рішень.

Таким чином, можна звернути увагу на кілька суперечливих моментів. Як наприклад, створення нових рішень пов’язано зі складною та тривалою розробкою та подальшим супроводом, складність яких напряму пов’язані з загальним рівнем розвитку сучасного програмного забезпечення. Також, звертаю увагу на відсутність спроб розвинути існуючі рішення шляхом їх об’єднання з іншими, що можна виправдати відносною легкістю, адже ці методики те технології вже перевірені часом і в наявності досвід їх використання. Впровадження їх модифікацій в такому плані викликає менші ризики, зокрема і при поєднання метрик складності програмного коду.

Виклад основного матеріалу

Сучасна практика розробки та аналізу програмного забезпечення володіє значною кількістю способів виміряти складність створеного алгоритму. Перш за все, використовуються показники часової складності (максимальна кількість часу для всіх можливих даних розміру n) та просторова складність (кількість додаткової пам’яті, яку необхідна при збільшенні об’єму вхідних даних) [7], які можна назвати базовими.

Окрім цих показників, отримали розповсюдженні різноманітні метрики, які розглядають складність програмного застосунку з точки зору різних аспектів. Так, розрізняють метрики трьох груп:

– розмірно-орієнтовані метрики – обчислюються на основі підрахунку певних характеристик вихідного коду. Характерні простотою використання;

– метрики складності потоку керування програми – напрямлені на дослідження різноманіття варіантів сценарії виконання програмного коду;

– метрики складності потоку керування даними – розглядають складність через використання вхідних даних, де складність прямо пропорційна частоті використання даних.

Таким чином, враховуючи такі підходи, ми пропонуємо модифікацію метрики складності потоку керування програми – цикломатична складність (інша назва - цикломатичне число МакКейба). Критерії, за допомогою яких вона оцінює складність потоку виконання програми, вираховуються на основі графу потоку керування. Шляхом розрахунків цикломатичну складність можна визначити за допомогою формули:

$$V(G) = e - n + 2p, \tag{1}$$

де e – кількість дуг, n – кількість вершин, p – компонент зв’язності.

Значення компонента зв’язності можна сприймати як кількість дуг, які необхідно додати для того, щоб граф потоку виконання програми став сильно зв’язаним, тобто коли дві його вершини є взаємно досяжними і з будь-якої вершини є шлях до будь-якої іншої вершини.

Для графів коректних програм, тобто для таких, які не містять недосяжних вершин від точки входу та з кожної вершини якої можна дістатись до кінцевої, сильна зв’язаність досягається шляхом з’єднання дугою кінцевої вершини та початкової. Можна стверджувати, що число V(G) визначає кількість лінійно незалежних контурів у сильно зв’язаному графі. Оскільки для коректної програми p=1, то формула набуває спрощеного виду:

$$V(G) = e - n + 2, \tag{2}$$

Як правило, при обчисленні цикломатичної складності логічні оператори не приймаються до уваги, допускається також спрощений підхід, згідно з яким власне побудова графа не проводиться, а показник

визначається на підставі підрахунку кількісті операторів керуючої логіки (if, switch і т. д.) і можливої кількості шляхів виконання програми. Дана метрика має кілька варіацій обчислення:

– «модифікована» цикломатична складність – розглядає не кожне розгалуження оператора множинного вибору (switch, case), а весь оператор як єдине ціле;

– «сувора» цикломатична складність – містить логічні оператори;

– «спрощена» цикломатична складність – передбачає обчислення на основі не графа, а підрахунку керуючих операторів;

– «актуальна» складність – визначається як кількість незалежних шляхів, що проходить код при тестуванні;

– метрика складності глобальних даних – обчислюється як цикломатична складність модуля та збільшується на кількість взаємозв'язків з глобальними даними.

Метрика цикломатичної складності може бути розрахована для модуля, методу та інших структурних одиниць програмного забезпечення.

Завдяки такій методиці розрахунку, цикломатична складність є метрикою, яка не залежить від мови програмування та результати обчислення котрої легко розуміти. Завдяки цьому, ця метрика стала дуже розповсюдженою і дозволяє легко створювати похідні метрики. Власне, тому ця метрика і була обрана.

Однак, дана метрика має серйозний недолік: цикломатична складність не враховує складність операцій. Таким чином отримати два різних програмних застосунки з приблизно однаковим значенням цикломатичної складності, проте справжня їх складність буде відрізнятися.

Тому нами запропоновано метрика диференційованої цикломатичної складності, яка дозволить здійснити об’єднання цикломатичної складності з метрикою, яка б могла подолати цей недолік.

Для цього нами були розглянуті метрики трьох зазначених груп. Метрики складності потоку керування програми, очевидно, не підходять, оскільки вони входять до однієї групи.

Серед розмірно-орієнтованих метрик розглянуто SLOC-метрику, її аналоги та метрику Холстеда. SLOC-метрика та її аналоги надають показники у вигляді кількісної характеристики коду (кількість рядків, операторів, методів тощо). Самі по собі такі показники вдається використати, але це окремі випадки. Метрика Холстеда представляє собою сукупність кількох показників, які послідовно розраховуються і формують набір показників, по яким вже можна оцінити програму. Для цієї метрики вже цикломатична складність є зайвою.

Серед метрик складності потоку керування даними були розглянуті: метрика Чепіна, пара «модуль–глобальна змінна», метрика Спена, метрика Кафура.

Метрика Чепіна розраховується на основі використовуваних змінних вводу-виводу окремого програмного модуля, розділяючи змінні на 4 групи, кожна з яких має власний ваговий коефіцієнт. Показником метрики виступає сума добутків кількості змінних кожної групи на власні коефіцієнти.

Пара «модуль-глобальна змінна» розраховує ймовірність посилання довільного модуля на довільну глобальну змінну в програмі. Висока вірогідність вказує на ризик «несанкціонованої» зміни будь-якої змінної, що може суттєво ускладнити роботи, пов'язані з модифікацією програмного застосунку.

Метрика Спена обчислює кількість тверджень, які містять один й той самий ідентифікатор. Великі показники спену вказують, що тестування та налагодження проходитиме складніше.

Метрика Кафура використовує поняття локального потоку даних, глобального потоку даних. Ця метрика обчислюється на основі інформаційної складністі процедур, що проходять через модуль програмного застосунку, та складності модуля відносно структури даних програмного застосунку.

В ідеалі, модифікація метрики цикломатичної складності має надати лише один додатковий показник, який би дозволив би відрізнити програми різної складності. Серед вказаних метрик була обрана метрика Чепіна через простоту отримання та розуміння показника.

Сама по собі метрика обчислюються за формулою:

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T, \quad (3)$$

де a_1, a_2, a_3, a_4 – вагові коефіцієнти,

P – кількість змінних, що вводяться, для розрахунків і для забезпечення виведення,

M – кількість змінних, які модифіковані або створювані всередині програмного застосунку,

C – кількість змінних, що беруть участь в управлінні програмним модулем (керуючі змінні),

T – кількість змінних, які не використовуються в програмному застосунку («паразитні»).

Згідно думки автора метрики, вагові коефіцієнти, становлять: a_1=1, a_2=2, a_3=3, a_4=0,5.

Таким чином, запропонована метрика диференційованої цикломатичної складності визначається наступним чином:

$$D(G) = [V(G), Q_\alpha], \quad (4)$$

де $V(G)$ – цикломатична складність,

Q_a – значення загальної середньої складність програмних модулів, яка визначається за формулою:

$$Q_a = \sum_{0}^{i} Q_i, \tag{5}$$

де Q_i – середня складність програмних модулів на i-тому шляху виконання програми:

$$Q_i = \frac{\sum_0^n Q_{ij}}{n}, \tag{6}$$

де Q_{ij} – значення метрики Чепіна для j-того програмного модуля на i-тому шляху виконання програмного застосунку,

n – кількість програмних модулів, з яких складається шлях виконання програмного застосунку.

Реалізацію обчислення диференційованої цикломатичної складності слід розглядати двома способами: виконання «вручну» та автоматизоване.

Виконання «вручну» мало чим відрізняється від збору інформації двох окремих метрик і його характер визначається варіацією обчислення цикломатичної складності. Тобто, наприклад, при обчисленні «модифікованої» цикломатичної складності, середня складність програмних модулів буде включати значення метрики Чепіна всіх програмних модулів, які включатимуть в собі оператори множинного вибору.

Для реалізації програмними засобами можна використати два підходи: повний та спрощений, по аналогії з метрикою цикломатичної складності. Повний підхід включає повноцінний аналіз коду з пошуком шляхів його виконання та побудування графу керування, де визначена належність кожного програмного модуля до свого шляху. Спрощений підхід обмежується лише підрахунком кількості операторів керуючої логіки, кількості програмних модулів та їх загальної середньої складності. Перший спосіб доречний у випадку дослідження коду на наявність занадто складних частин, де загальний аналіз нічого не дасть.

Запропонована метрика дозволяє краще визначити рівень складність програмного застосунку, код яких може мати приблизно однакову кількість способів виконання. Нова метрика може бути використана на етапі проектуванні для завчасного визначення теоретичної складності програмного додатку та на етапі тестування для безпосереднього визначення складності розроблюваного програмного забезпечення. Метрика дозволяє проводити аналіз програмних застосунків з різноманітними структурами, зокрема – ефективно себе демонструє при аналізі додатків з багаторівневою структурою, зокрема, які працюють з базами даних та системами керування базами даних.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

У цій роботі було розглянуто частино робіт у області аналізу програмного коду. Досліджені роботи демонструють різносторонні підходи до покращення процесу аналізу у межах досягнення різноманітних результатів, але не спостерігалися спроби використання методик, побудованих на основі вже існуючих та перевірених рішень.

Була запропонована метрика диференційованої цикломатичної складності, створена для усунення недоліку оригінальної метрики цикломатичної складності у вигляді нездатності розрізнити складність двох програмних застосунків з однаковою кількістю шляхів. Утворена метрика дозволяє визначити, за допомогою використання показників метрики Чепіна, складність операцій, що відбувається всередині алгоритму, що дозволяє розрізняти два програмних застосунки різної реальної складності з однакової кількістю шляхів виконання. Тобто, диференційованої цикломатична складність демонструє більшу точність у порівнянні з оригінальною метрикою. У тому числі, при аналізі програмних застосунків з різноманітними структурами, зокрема – ефективно себе демонструє при аналізі програмних застосунків, які працюють з базами даних, системами керування базами даних та програмних застосунків з багаторівневою структурою зокрема.

References

1. Salma Eid, Soha Makady, Manal Ismail, Detecting software performance problems using source code analysis techniques, Egyptian Informatics Journal, Volume 21, Issue 4, 2020, Pages 219-229, ISSN 1110-8665, https://doi.org/10.1016/j.eij.2020.02.002.
2. Chen Tsfaty, Michael Fire, Malicious source code detection using a translation model, Patterns, Volume 4, Issue 7, 2023, 100773, ISSN 2666-3899, https://doi.org/10.1016/j.patter.2023.100773.
3. Panchanan Nath, Jaya Rani Mushahary, Ujjal Roy, Maharaj Brahma, Pranav Kumar Singh, AI and Blockchain-based source code vulnerability detection and prevention system for multiparty software development, Computers and Electrical Engineering, Volume 106, 2023, 108607, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2023.108607.
4. Gholamali Nejad Hajali Irani, Habib Izadkhah, Sahand, 1.0: A new model for extracting information from source code in object-oriented projects, Computer Standards & Interfaces, 2023, 103797, ISSN 0920-5489, https://doi.org/10.1016/j.csi.2023.103797.
5. Gordana Rakić, Melinda Tóth, Zoran Budimac, Toward recursion aware complexity metrics, Information and Software Technology, Volume 118, 2020, 106203, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2019.106203.

6. Zixuan Song, Hui Zeng, Xiuwei Shang, Guanxi Li, Hui Li, Shikai Guo, An data augmentation method for source code summarization, Neurocomputing, Volume 549, 2023, 126385, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2023.126385.
7. Savchuk V. Big O: Skladnist algoritmiv [Internet-resurs]. – Rezhym dostupu: https://www.the-code.com.ua/skladnist-alghoritmiv/vilniy.