УДК 004.415.2.045 DOI: 10.18372/2073-4751.85.21099

І. Е. Райчев, к.т.н,
https://orcid.org/0000-0003-4058-1236,
ihor.raichev@npp.kai.edu.ua;
А. А. Хрустовський
https://orcid.org/0009-0001-0497-4836,
7496877@stud.kai.edu.ua;

ТЕХНОЛОГІЯ ОЦІНЮВАННЯ ЯКОСТІ КОМПОНЕНТІВ ПРОГРАМНИХ СИСТЕМ НА БАЗІ МЕТРИК СКЛАДНОСТІ КОДУ

Державний університет «Київський авіаційний інститут»

Вступ

В процесі створення програмного забезпечення (ПЗ), дуже часто виникає проблема: код працює, але підтримувати його досить складно. Структура заплутана, логіка незрозуміла, а вносити зміни без того, щоб щось не зламати, майже неможливо. Традиційний підхід, який полягає у перевірці якості вже після того, як програмну систему (ПС) зібрано і протестовано, – не завжди допомагає. До цього моменту проблеми вже накопичилися, а виправляти їх є занадто трудомістко й витратно.

Для вирішення цієї проблеми використаємо метричну систему М.Холстеда, метрики Т. Маккейба та стандарти серії SQuaRE [1–3]. ISO/IEC 25010 описує модель якості програмного продукту, а ISO/IEC 25023 пропонує конкретні метрики для для отримання мір оцінки характеристик якості. Одна з ключових характеристик якості — це супроводжуваність (maintainability), яка показує, наскільки легко можна змінювати, виправляти чи вдосконалювати ПЗ [1; 2].

Супроводжуваність складається з кількох підхарактеристик: модульність, можливість багаторазового використання, аналізованість, модифікованість і тестованість [2;4]. Останні три: аналізованість, модифікованість і тестованість — безпосередньо залежать від того, як написаний код програми. Тому тут доречно застосувати метрики складності.

Метрики М. Холстеда та Т. Маккейба — це класичні способи оцінити складність програми ще до її виконання, виконавши аналіз вихідного коду [5; 6]. Холстед підраховує оператори й операнди та на їх основі обчислює "інформаційну насиченість" коду [5; 7]. Маккейб підраховує кількість незалежних шляхів виконання алгоритму програми, що називається її цикломатичною складністю [6; 8].

Чому це важливо? Якщо метрики мають високі значення, це сигнал, що код складний, його важко розуміти, тестувати і змінювати. А значить, *супроводжуваність* під загрозою.

Мета статті — показати, як метрики якості програмного забезпечення, зокрема метрики М. Холстеда та Т. Маккейба, можна використати для оцінювання внутрішньої якості реального коду, відповідно до характеристик та підхарактеристик, поданих у стандарті ISO/IEC 25010.

Аналіз останніх досліджень та постановка проблеми

Питання кількісного оцінювання складності коду розглядається в науковій літературі вже декілька десятиліть. Метрики М.Холстеда запропоновано у 1977р., цикломатичну складність Т.Маккейба — у 1976р. Обидві методики і досі активно застосовуються в інструментах статичного аналізу — від Klocwork до SonarQube [5–8].

Зв'язок між цими метриками і підтримуваністю коду досліджувався у різних контекстах. Зокрема, у роботі з Repository UB метрики Холстеда і Маккейба поєднують для обчислення Maintainability Index — числа від 0 до 100, яке показує, наскільки легко підтримувати модулі [9]. Практичні порогові значення для цикломатичної складності ($M \leq 7$ – це прийнятно, $M > 10$ – це сигнал для початку рефакторингу) задокументовані в технічній документації кількох інструментів [8].

Стандарти ISO/IEC 25010 та ISO/IEC 25023 при цьому описують модель якості і систему мір, але жорстко не прив’язують конкретні метрики до конкретних підхарактеристик, що залишається на розсуд дослідника [1–3]. Невирішеною залишається задача систематизованої інтерпретації результатів статичного аналізу коду в координатах моделі якості SQuaRE, яка розглядається у даній роботі.

Стандарти групи SQuaRE і супроводжуваність програм

Стандарти серії SQuaRE (Systems and software Quality Requirements and Evaluation) — це, по суті, рамка та базис формалізації вимог до якості ПЗ (див. рис.1). ISO/IEC 25010 задає модель якості продукту, а ISO/IEC 25023 пропонує конкретні метрики і міри для оцінки [1–3].

У моделі ISO/IEC 25010 якість програмного продукту описується через множину характеристик якості. Супроводжуваність — одна з них. Вона означає, наскільки ефективно можна модифікувати систему, виправляти дефекти, адаптувати ПС до нового середовища або додавати до компонентів нову функціональність [2; 10].

Це критично важливо, бо більша частина витрат на розробку ПС припадає саме на етап супроводу. Не на початкову розробку, а на підтримку та зміни у ПЗ.

Супроводжуваність включає п'ять підхарактеристик (табл. 1).

Для статичного аналізу найбільш релевантні підхарактеристики аналізованість, модифікованість і тестованість, бо саме вони найтісніше пов’язані зі структурою коду ПС.

Таблиця 1. Підхарактеристики супроводжуваності в ISO/IEC 25010

Підхарактеристика	*Зміст*
Модульність	Відображає здатність програмної системи бути організованою у відносно незалежні компоненти, де зміни в одному компоненті мінімально впливають на інші.
Можливість багаторазового використання	Характеризує можливість використання активу коду більш ніж в одній системі або підсистемі.

Підхарактеристика	*Зміст*
Аналізованість	Показує, наскільки ефективно можна оцінити вплив змін або визначити причини дефектів у ПЗ.
Здатність до модифікації	Описує можливість змінювати систему без суттєвого зниження її якості.
Тестованість	Відображає здатність системи підтримувати встановлення критеріїв тестування та виконання перевірки.

З погляду внутрішньої якості саме структура програмного коду є джерелом багатьох проблем. Ось що робить код важким для супроводу:

- надто багато вкладених if-else;
- купа умовних переходів;
- довгі методи, де все намішано;
- велика кількість операторів і змінних.

Усе це ускладнює розуміння логіки програми і підвищує ймовірність помилок під час внесення змін [2; 6].

Саме тому внутрішні міри якості, які можна обчислити на етапі статичного аналізу (тобто без запуску програми), є такими корисними. Вони допомагають виявити проблеми ще до того, як код потрапив у production [1; 11].

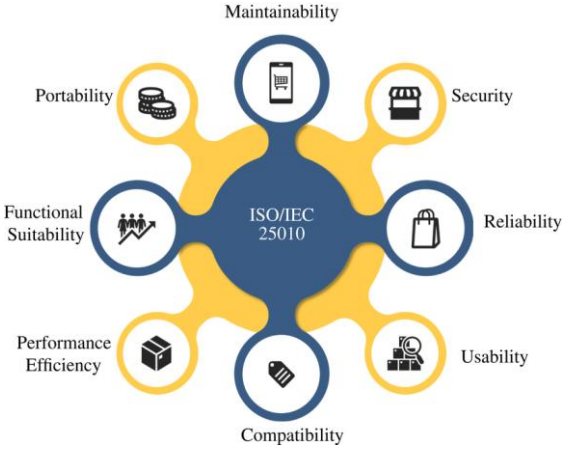

Рис. 1. Місце супроводжуваності (maintainability) в моделі якості SQuaRE

Метрики М. Холстеда

Метрики Холстеда — один із класичних методів оцінки складності [5;7]. Ідея є простою: розділити код на оператори (operators) і операнди (operands), порахувати їх та на основі цього обчислити декілька показників.

Базові величини:

- n_1 — кількість різних (унікальних) операторів;
- n_2 — кількість різних операндів;
- N_1 — загальна кількість всіх операторів у коді;
- N_2 — загальна кількість всіх операндів.

Що таке оператори й операнди?

- Оператори: +, - , *, /, =, if, for, while, виклики функцій тощо.
- Операнди: змінні, константи, числа — все, з чим оператори працюють.

На основі цих чотирьох чисел обчислюються такі метрики:

- Словник програми(Vocabulary):$n=n_1+n_2$.
- Довжина програми (Length): $N=N_1+N_2$.
- Обсяг (Volume): $V = N \times \log_2(n)$.
- Складність(Difficulty): $D = (n_1/2) \times (N_2/n_2)$.
- Зусилля (Effort): $E = V \times D$.

Обсяг показує інформаційну насиченість коду, а складність і зусилля

— наскільки важко розуміти і змінювати код. Тому метрики Холстеда корисні для оцінки аналізованості та модифікованості.

Таблиця 2. Основні метрики М. Холстеда

Метрика	*Формула*	*Що показує*
Vocabu-lary	$n = n_1 + n_2$	Різноманітність елементів коду
Length	$N = N_1+N_2$	Загальний обсяг операторів і операндів коду
Volume	$V = N \times \log_2(n)$	Інформаційна насиченість
Difficulty	$D = (n_1/2) \times (N_2/n_2)$	Складність розуміння та модифікації
Effort	$E = V \times D$	Умовна оцінка зусиль на опрацювання

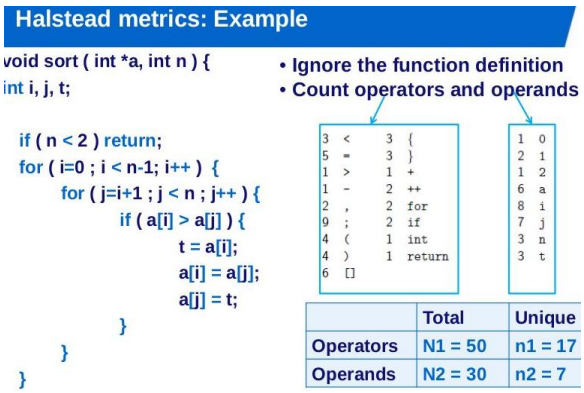

Рис. 2. Класифікація операторів і операндів для обчислення метрик Холстеда.

Цикломатична складність

Цикломатична складність Маккейба — це кількість незалежних шляхів виконання програми [6; 8].

Чим більше if, for, while, case — тим більше шляхів, і тим складніша логіка.

Формула складності: M = E - N + 2, де E — кількість ребер графа потоку керування; N — кількість вузлів графа.

На практиці для довільної функції це можна порахувати так [8]:

M = 1 + кількість if + кількість циклів + кількість case. Наприклад, якщо у функції є один if та один цикл, то M = 1+1+1 = 3.

Чим вище цикломатична складність програми, тим:

- важче побудувати повний набір тестів (бо потрібно покрити всі шляхи виконання);
- важче зрозуміти, що робить функція;
- більше ризик прихованих багів у складних умовах коду.

Тому метрика Маккейба тісно пов'язана з тестованістю та аналізованістю [6;7;9].

Таблиця 3. Цикломатична складність

Метрика	*Формула*	*Що показує*
Cyclomatic Complexity	M = E-N+2, або M = 1+ifs + loops+cases	Кількість незалежних шляхів виконання. Чим їх більше, тим складніше тестувати.

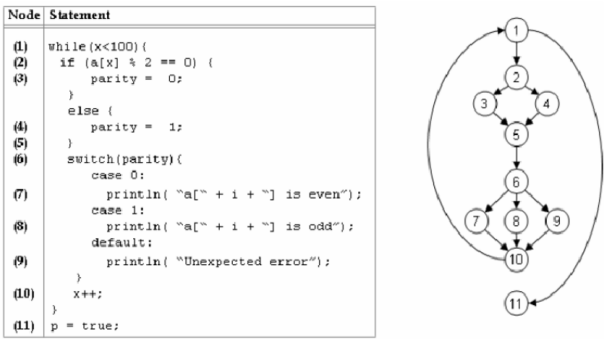

Рис. 3. Граф потоку керування для обчислення цикломатичної складності Маккейба

Практичний приклад: аналіз реального коду

Щоб показати, як це працює на практиці, проаналізуємо реальну Python-функцію:

```
def calculate_discount(price, customer_type, quantity):
  if price <= 0:
    return 0

  discount = 0
  if customer_type == "regular":
    if quantity >= 10:
      discount = 0.05
    elif quantity >= 5:
      discount = 0.03
  elif customer_type == "premium":
    if quantity >= 10:
      discount = 0.15
    elif quantity >= 5:
      discount = 0.10
    else:
      discount = 0.05

  final_price = price - (price * discount)
  return final_price
```

Ця функція розраховує ціну зі знижкою залежно від типу клієнта (regular або premium) і кількості товару.

Обчислення метрик Холстеда

Спочатку виділимо оператори й операнди. Оператори ($n_1 = 10$ різних):
1. def, 2. if, 3. <=, 4. return, 5. =, 6. ==, 7. >=, 8. elif, 9. -, 10. *

Підрахунок загальної кількості операторів (N_1). Загалом: $N_1 \approx 28$.

Операнди ($n_2 = 15$ різних):
1. calculate_discount, 2. price, 3.customer_type, 4. quantity, 5. 0, 6. discount, 7. "regular", 8. 10, 9. 0.05, 10. 5, 11. 0.03, 12. "premium", 13. 0.15, 14. 0.10, 15. final_price .

Підрахунок загальної кількості операндів (N_2). Загалом: $N_2 \approx 45$.

Результати обчислень:

- Словник: $n = n_1 + n_2 = 10 + 15 = 25$.
- Довжина: $N = N_1 + N_2 = 28 + 45 = 73$.
- Обсяг: $V = N \times \log_2(n) = 73 \times \log_2(25) \approx 73 \times 4.64 \approx 339$.
- Складність: $D = (n_1/2) \times (N_2/n_2) = (10/2) \times (45/15) = 5 \times 3 = 15$.
- Зусилля: $E = V \times D = 339 \times 15 \approx 5085$.

Таблиця 4. Вихідні дані для обчислення метрик Холстеда

Показник	*Значення*	*Пояснення*
n_1	10	Кількість різних операторів (def, if, ==, return тощо).
n_2	15	Кількість різних операндів (змінні, константи).
N_1	28	Загальна кількість операторів коду.
N_2	45	Загальна кількість всіх операндів коду.

Таблиця 5. Результати обчислення метрик Холстеда

Метрика	*Значення*	*Інтерпретація*
Vocabulary	25	Помірна різноманітність елементів.
Length	73	Невеликий фрагмент коду.
Volume	339	Код має помітну інформаційну насиченість.
Difficulty	15	Розуміння та модифікація потребують помірних зусиль.
Effort	5085	Узагальнена оцінка трудомісткості.

Обчислення цикломатичної складності Маккейба

Підрахуємо кількість точок прийняття рішень: 7 умов (if/elif). За формулою: M = 1 + 7 = 8. Цикломатична складність 8 означає, що є 8 незалежних шляхів виконання. Це вже не "проста" функція — для повного покриття тестами потрібно продумати всі 8 тестових сценаріїв.

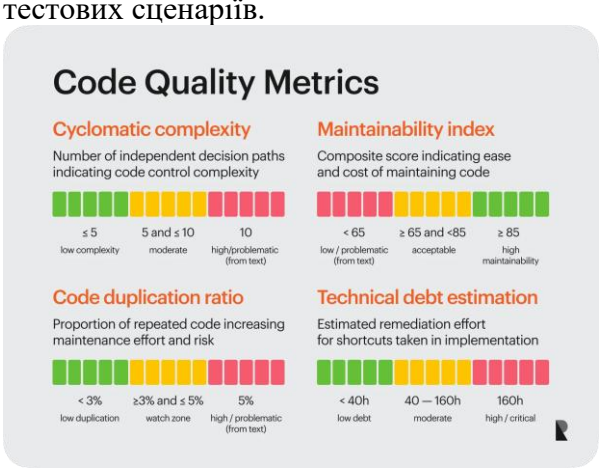

Рис. 4. Послідовність обчислення метрик внутрішньої якості коду.

Інтерпретація результатів у контексті SQuaRE

Отримані значення метрик потрібно інтерпретувати не просто як "добре" чи "погано", а в контексті підхарактеристик якості стандарту SQuaRE.

Цикломатична складність Маккейба (M=8), що безпосередньо пов'язано з тестованістю. Щоб повністю протестувати функцію, потрібно покрити 8 різних шляхів виконання. Це вже непросто — треба придумати тестові кейси для кожної комбінації умов.

Також це впливає на аналізованість: велика кількість розгалужень робить поведінку функції менш очевидною. Щоб зрозуміти, що станеться для конкретних значень price, customer_type та quantity, доведеться уважно простежити логіку.

Метрики Холстеда (Volume = 339, Difficulty = 15, Effort = 5085). Ці показники пов'язані з аналізованістю та модифікованістю. Обсяг 339 — це помірна інформаційна насиченість. Складність 15 і зусилля 5085 показують, що для розуміння та модифікації цього коду потрібні певні зусилля.

Якщо порівняти кілька функцій і знайти ту, де ці показники значно вищі, — це сигнал, що функція потребує рефакторингу або спрощення.

Таблиця 6. Відповідність метрик складності підхарактеристикам якості

Метрика	*Підхарактеристика якості*	*Пояснення*
Cyclomatic Complexity	Тестованість	Більше шляхів виконання означає більше тестових сценаріїв.
Cyclomatic Complexity	Аналізованість	Багато розгалужень знижують прозорість логіки.

Метрика	*Підхарактеристика якості*	*Пояснення*
Halstead Volume	Аналізованість	Висока інформаційна насиченість ускладнює сприйняття.
Halstead Difficulty	Здатність до модифікації	Складніший код важче змінювати без помилок.
Halstead Effort	Супроводжуваність (загалом)	Умовна оцінка зусиль на розуміння та опрацювання коду.

Важливо розуміти, що жодна окрема метрика не дає повної картини якості ПЗ. Характеристики якості функціональна придатність, ефективність виконання, безпека, зручність використання не можуть бути виміряні лише аналізом коду. Тому метрики Холстеда і Маккейба слід використовувати як частину ширшого процесу оцінювання, поєднуючи їх з динамічним тестуванням і кваліфікованими експертними оцінками.

Висновки

В результаті проведеного дослідження можна констатувати, що метрики складності коду — це корисний інструмент для оцінювання внутрішньої якості програмного забезпечення на базі стандартів SQuaRE.

Стандарти ISO/IEC 25010 та ISO/IEC 25023 [3; 12] дають методологічну основу для системного підходу до оцінки якості. Супроводжуваність як ключова характеристика [13] безпосередньо залежить від структури коду, а тому застосування статичних метрик в процесі оцінювання якості доречне та ефективне, тим більше, що метрики ISO/IEC 25023 обчислюються в основному для «працюючого» ПЗ.

Метрики Холстеда та Маккейба [8;14] надають можливість отримати конкретні міри, тобто числа, які можна інтерпретувати як значення якості підхарактеристик супроводжуваності.

Цикломатична складність Маккейба пов'язана з тестованістю (скільки тестів потрібно), аналізованістю (наскільки складна логіка) та модульністю (наскільки складний код кожного модулю).

Метрики Холстеда відображають загальну оцінку, а також аналізованість (скільки інформації треба опрацювати) і здатність до модифікації (наскільки важко змінювати код компонентів).

На практиці міри метрик дозволяють виявити проблемні фрагменти коду ще на ранніх етапах розробки ПС , що значно знижує вартість супроводу і підвищує загальну якість програмного продукту.

У наведеному прикладі з Python-функцією calculate_discount ми отримали:

Обсяг Холстеда ≈ 339, зусилля ≈ 5085 — код помірно складний для сприйняття.

Цикломатична складність дорівнює 8, тобто потрібно щонайменше 8 тестових сценаріїв для повного покриття логіки.

Це реальні цифри, які допомагають прийняти рішення: чи потрібен рефакторинг, чи достатньо тестів, а також де можуть бути приховані баги.

Але метрики складності — це лише частина комплексної оцінки якості. Вони не показують наскільки програма функціонально правильна, безпечна та зручна. Такі метрики потрібно використовувати разом з іншими методами: code review, динамічним тестуванням, аналізом зовнішньої якості.

Перспективи подальших досліджень полягають у розробці інтегрованих моделей, які поєднують статичні метрики з показниками функціональності, ефективності

виконання, безпеки та зручності використання [15].

Література

1. ISO/IEC 25010:2023. Systems and software engineering — Systems and software quality models. Geneva : International Organization for Standardization, 2023. 45 p.
2. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : International Organization for Standardization, 2011. 34 p.
3. ISO/IEC 25023:2016. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality. Geneva : International Organization for Standardization, 2016. 52 p.
4. Maintainability [Електронний ресурс] / ISO 25000. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/57-maintainability (дата звернення: 11.03.2026).
5. Halstead Metrics [Електронний ресурс] / Verifysoft Technology. URL: https://www.verifysoft.com/en_halstead_metrics.html (дата звернення: 11.03.2026).
6. What is Cyclomatic Complexity [Електронний ресурс] / Qt Quality Assurance Blog. 2018. URL: https://www.qt.io/quality-assurance/blog/what-is-cyclomatic-complexity (дата звернення: 11.03.2026).
7. Halstead [Електронний ресурс] / Objectscript Quality Documentation. URL: https://objectscriptquality.com/docs/metrics/halstead (дата звернення: 11.03.2026).
8. McCabe Cyclomatic Complexity [Електронний ресурс] / Klocwork Documentation. 2025. URL: https://help.klocwork.com/current/en-us/concepts/mccabecyclomaticcomplexity.htm (дата звернення: 11.03.2026).
9. Pembangunan Kakas Bantu Untuk Mengukur Maintainability Index Pada Perangkat Lunak Berdasarkan Nilai Halstead Metrics Dan McCabe's Cyclomatic Complexity [Електронний ресурс] / Repository UB. 2019.
10. ISO/IEC 25010 [Електронний ресурс] / ISO 25000. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (дата звернення: 11.03.2026).
11. Code Complexity Metrics: Writing Clean, Maintainable Software [Електронний ресурс] / Iterators HQ. 2025. URL: https://www.iteratorshq.com/blog/code-complexity-metrics-writing-clean-maintainable-software/ (дата звернення: 11.03.2026).
12. ISO 25010 Standard [Електронний ресурс] / Grounded Architecture. 2021. URL: https://grounded-architecture.io/iso25010 (дата звернення: 11.03.2026).
13. Achieving Maintainability with ISO/IEC 25010:2023 [Електронний ресурс] / QMII. 2022. URL: https://www.qmii.com/achieving-maintainability-with-iso-iec-250102023/ (дата звернення: 11.03.2026).
14. Simulink Halstead Complexity [Електронний ресурс] / MathWorks Documentation. 2024. URL: https://www.mathworks.com/help/slcheck/ref/simulink-operators-and-operands.html (дата звернення: 11.03.2026).
15. ISO/IEC 25010 – Systems and Software Quality [Електронний ресурс] / Quality Arc42. 2026. URL: https://quality.arc42.org/standards/iso-25010 (дата звернення: 11.03.2026).

ТЕХНОЛОГІЯ ОЦІНЮВАННЯ ЯКОСТІ КОМПОНЕНТІВ ПРОГРАМНИХ СИСТЕМ НА БАЗІ МЕТРИК СКЛАДНОСТІ КОДУ

У статті досліджені методи та засоби оцінювання внутрішньої якості програмного забезпечення з використанням метрик складності коду за М. Холстедом та Т. Маккейбом. Виконано загальний аналіз характеристики якості супроводжуваність та її підхарактеристик зі стандарту SQuaRE ISO/IEC25010, які можна ефективно застосувати для оцінки супроводжуваності програмних систем. Основну увагу приділено метрикам Холстеда та Маккейба, оскільки вони дають змогу кількісно оцінити структурну складність коду програмних компонентів. Показано, що ці метрики корисні та ефективні для визначення мір якості аналізованості, модифікованості та тестованості програмних модулів. Запропоновано підхід до інтерпретації результатів статичного аналізу коду на основі моделі якості стандарту SQuaRE. Практичну частину проілюстровано на прикладі Python-коду.

Ключові слова: *якість програмного забезпечення, внутрішня якість, SQuaRE, ISO/IEC 25010, супроводжуваність, метрики Холстеда, цикломатична складність Маккейба, статичний аналіз коду.*

Raichev I., Khrustovskyi A.

TECHNOLOGY FOR ASSESSING THE QUALITY OF PROGRAM SYSTEMS COMPONENTS BASED ON CODE COMPLEXITY METRICS

The article investigates methods and tools for assessing the internal quality of software using code complexity metrics according to M. Halstead and T. McCabe. A general analysis of the maintainability quality characteristic and its subcharacteristics from the SQuaRE ISO/IEC 25010 standard is performed, which can be effectively used to assess the maintainability of program systems. The main attention is paid to the Halstead and McCabe metrics, since they allow quantitatively assessing the structural complexity of the code of software components. It is shown that these metrics are useful and effective for determining the quality measures of analyzability, modifiability and testability of software modules. An approach to interpreting the results of static code analysis based on the SQuaRE quality model is proposed. The practical part is illustrated by an example of Python code.

Keywords*: software quality, internal quality, SQuaRE, ISO/IEC 25010, maintainability, Halstead metrics, McCabe cyclomatic complexity, static code analysis.*

Стаття подана до редакції: 23/03/2026
Стаття прийнята до опублікування: 30/03/2026
Стаття опублікована: 27/04/2026

Стаття поширюється на умовах ліцензії CC BY 4.0