

УДК 004.413:681.5

О. О. Дяченко, О. В. Грабовський, к.т.н.

Державний університет інтелектуальних технологій і зв'язку

МЕТРИКИ ЯКОСТІ КОДУ ЯК ІНСТРУМЕНТ УДОСКОНАЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНО-ВИМІРЮВАЛЬНИХ СИСТЕМ

Стаття присвячена ролі метрик якості коду як інструменту підвищення надійності та експлуатаційної придатності програмного забезпечення інформаційно-вимірювальних систем. Дослідження спирається на екосистему відкритих інструментів (SonarQube, Jenkins, JUnit, Radon, Infer) і демонструє, як їх інтеграція у CI/CD-процеси переводить контроль якості з рівня локальних практик на рівень відтворюваної інженерної процедури. Теоретичну основу становить уявлення про якість як багатовимірну конструкцію, що поєднує структурну простоту коду (цикломатична складність), достатність верифікації (покриття тестами), боргове навантаження (технічний борг), часову ефективність (латентність/пропускна здатність) і енергоефективність (споживання ресурсів на одиницю функціоналу). Саме ці компоненти безпосередньо корелюють з надійністю, стабільністю та безпекою критичних програмних контурів. Результати показують, як автоматизований метод дозволяє адаптувати програмне забезпечення інформаційних та вимірювальних систем до сучасних технологічних викликів, зокрема до поєднання хмарних обчислень, штучного інтелекту та вимог енергоефективності.

Ключові слова: якість програмного забезпечення, метрики програмного забезпечення, якість коду, показники програмного забезпечення, управління якістю відкритого програмного забезпечення.

О. О. Dyachenko, O.V. Grabovskii, PhD

CODE QUALITY METRICS AS A TOOL FOR IMPROVING THE SOFTWARE OF INFORMATION AND MEASUREMENT SYSTEMS

In the fields of business, medicine, telecommunications, and research, modern information and measurement systems (IMS) are essential, and software is vital to guaranteeing precision, dependability, and effectiveness. The safety, performance, and sustainability of numerous systems are directly impacted by their quality in 2025, the year of technological advancements including the integration of artificial intelligence (AI), the use of cloud computing, and energy efficiency. The purpose of the article. The goal of this research is to show how code quality metrics, supported by automated open-source tools such as SonarQube, Jenkins, and JUnit, could increase the effectiveness of TDF software and go beyond traditional approaches to assessment. The main objective of the research is to analyze and improve software in order to guarantee that it complies with strict requirements in real time. The research is aimed at analyzing and improving software to ensure its compliance with high standards in real time. Scientific novelty. The article extends the approach to code quality assessment proposed in PeerDH by adapting metrics to the specifics of TDFs, considering technological trends of 2025, such as AI integration, cloud computing, and energy efficiency. In contrast to the general principles discussed in previous studies, we analyze the practical application of metrics (cyclomatic complexity, test coverage, technical debt, execution time) to real systems (SmartGrid, MediScan, Industry 4.0) using open tools. The novelty lies in the development of an algorithm for integrating metrics into CI/CD processes and the emphasis on energy efficiency of code for IoT components of IPS.

Results. The study has shown that automated analysis of code quality metrics using SonarQube, Jenkins, and JUnit allows to identify weaknesses in IPS software, increase its reliability, and optimize performance. For SmartGrid, 92% test coverage and 45 ms execution time were achieved, for MediScan, code complexity was 5 and 98% coverage, and for Industry 4.0, the need for refactoring was identified due to complexity of 13 and technical debt of 160 hours. Practical recommendations are offered: Jenkins setup for CI/CD, SonarQube analysis with predefined Quality Gates, and JUnit testing for early detection of defects.

Keywords: software quality, software metrics, code quality, software metrics, open-source quality management.

DOI 10.32684/2412-5288-2025-2-27-116-127

Вступ. Наукові дослідження, промисловість, охорона здоров'я та телекомунікації в значній мірі покладаються на сучасні інформаційні та вимірювальні системи (ISM). Програмне забезпечення, яке управляє цими системами, повинно відповідати високим стандартам ефективності, надійності та обслуговування. Застосування метрик якості коду, які дозволяють оцінити структуру програмного продукту, виявити можливі проблеми та гарантувати його довгострокову життєздатність, є однією з основних областей вдосконалення програмного забезпечення.

У таких галузях, як охорона здоров'я, енергетика, виробництво та транспорт, інформаційні та вимірювальні системи (ISM) необхідні для забезпечення точного збору, обробки та аналізу даних у режимі реального часу. Якість програмного забезпечення цих систем безпосередньо впливає на їхню функціональність, безпеку та продуктивність. З огляду на стрімкий розвиток технологій, особливо в 2025 році, коли інтеграція штучного інтелекту, хмарних обчислень та автоматизованих процедур стане стандартом, вимірювання якості коду набуває важливого значення. Ця робота зосереджена на використанні автоматизованих інструментів з відкритим кодом (SonarQube, Jenkins, JUnit) для оцінки показників якості коду в IBS. Сучасний стан інформаційних та вимірювальних систем вимагає нових підходів до безпеки. Традиційні підходи до захисту зазвичай виявляються неефективними проти сучасних кіберзагроз, тому вимірювання якості коду має бути інтегровано в процес розробки. Це особливо актуально для систем, що управляють критичними даними в режимі реального часу.

Аналіз досліджень та публікацій. Показники якості коду є корисним інструментом для аналізу та вдосконалення програмного забезпечення, особливо в інформаційних та вимірювальних системах, де точність, надійність та ефективність мають вирішальне значення.

У сучасних дослідженнях інтеграція теорій інженерії інформаційних систем і програмного забезпечення для створення інструментів оцінки якості є актуальною темою. Almetwaly, A. A. I., та Fadhel, I. E. I. [1] пропонують підхід до вимірювання якості, який може бути адаптований до інформаційно-вимірювальних систем, де точність даних є критичною. Shah, H. M. та співавтори [2] фокусуються на об'єднанні метрик продуктивності та якості коду, що може бути корисним для масштабних інформаційних систем, де необхідно балансувати між швидкістю розробки та надійністю. Setiadi, D. та співавтори [3] аналізують метрики якості в академічних

інформаційних системах, методи оцінки яких можуть бути застосовані для забезпечення стабільності вимірювальних систем. Beningo, J. [4] розглядає метрики якості вбудованого програмного забезпечення, що є актуальним для інформаційно-вимірювальних систем, де часто використовуються вбудовані компоненти. Sortwell, O., Cutting, D., та McConnellogue, C. [5] аналізують автоматизовану оцінку якості коду через метрики, що може бути використано для оптимізації процесів у вимірювальних системах. Kodali, H. та співавтори [6] пропонують інструмент CodeGuardian для аналізу якості в рамках CI/CD, який дає змогу аналізувати якість коду в рамках CI/CD, що сприяє підвищенню надійності програмного забезпечення вимірювальних систем. Witte, F. [7] розглядає метрики для звітності про тестування, що є корисним для оцінки результатів у системах, де точність має критичне значення. Cruz-Lemus, J. A. та співавтори [8] присвячують свою статтю метрикам якості квантового програмного забезпечення, ідеї яких можуть бути адаптовані для вимірювальних систем.

Міжнародний стандарт ISO/IEC 25010:2023 [9] визначає модель якості програмного забезпечення, яка є ключовою основою для розробки та застосування метрик якості коду в численних дослідженнях і практичних проєктах. Ця модель структурує оцінку програмного забезпечення через вісім основних характеристик якості — функціональну придатність, продуктивність, сумісність, зручність використання, надійність, безпеку, ремонтпридатність і портативність, — кожна з яких може бути виміряна за допомогою специфічних метрик. Зокрема, такі метрики, як відсоток коду, який покривається тестами, рівень дефектів або час відгуку системи, дозволяють кількісно оцінити надійність і продуктивність програмного продукту. Завдяки своїй повній структурі стандарт дуже поширений в інформаційних та вимірювальних системах, пропонуючи розробникам і тестувальникам чіткі критерії для виявлення слабких місць коду, для підвищення його стабільності та продуктивності нарівні з критеріями безпеки та типовими критеріями.

Метод тестування, розроблений Олійником П. та Мартинюком В., підвищує ефективність перевірки коду та покращує якість даних і систем вимірювання [10]. Бармак О. В. та його команда [11] використовують метрику Халстеда для оцінки якості коду. Це хороший спосіб виміряти складність програмного забезпечення в системах вимірювання. Ця метрика використовує безрозмірну шкалу та кількісні показники, такі як розмір програми, рівень складності та

зусилля на розробку, щоб визначити ступінь складності.

У джерелах представлені різні способи вимірювання якості коду, які охоплюють як стандарти якості ISO/IEC 25010, так і окремі методології оцінювання, такі як метрики Холстеда та принципи квантового ядра. У контексті інформаційно-вимірювальних систем ключовими є точність, надійність і автоматизація оцінки якості програмного забезпечення. Зокрема, роботи Олійника П. та Бармака О.В. пропонують практичні рішення, які можуть бути інтегровані в такі системи [10].

Метою роботи є: розробка та впровадження автоматизованого підходу до використання метрик якості коду (цикломатична складність, покриття тестами, технічна заборгованість) за допомогою відкритих інструментів (SonarQube, Jenkins, JUnit) для підвищення надійності, безпеки та енергоефективності програмного забезпечення інформаційно-вимірювальних систем (ІВС). Для дослідження було обрано комбінований підхід, що включає: аналіз наукових публікацій з метрик якості коду, експериментальну перевірку на реальних ІВС, порівняльну оцінку безкоштовних інструментів аналізу (SonarQube, Radon, Jenkins).

Виклад основного матеріалу. Метрики якості коду це числові показники, які характеризують різні аспекти програмного продукту, такі як читабельність, підтримуваність, складність, надійність та ефективність. Їх отримують шляхом аналізу вихідного коду за допомогою спеціалізованих інструментів (наприклад, SonarQube, CodeGuardian або статичних аналізаторів), які вимірюють такі параметри, як кількість операторів, глибина вкладеності, цикломатична складність або частота помилок. Визначаються ці метрики на основі стандартів, наприклад, ISO/IEC 25010, або специфічних методик, таких як метрика Холстеда, що оцінює обсяг і складність програми. Вони є інструментом для оцінки стану ПЗ та допомагають приймати обґрунтовані рішення щодо його покращення. Як зазначено в I. Almetwaly A.A.I, Fadhel IEI [1, с. 3382], інженерія інформаційних систем та інженерія програмного забезпечення працюють разом, щоб надати корисні інструменти для поліпшення результатів шляхом вимірювання якості програмного забезпечення. Технічно це поєднання реалізується через застосування спільних методологій, таких як моделі оцінки якості (наприклад, ISO/IEC 25010), автоматизовані інструменти аналізу коду (SonarQube, CodeGuardian) та інтеграцію даних із інформаційних систем у процеси розробки, що призводить до

зниження дефектів, оптимізації продуктивності та полегшення супроводу програмного забезпечення.

У практиці оцінювання якості програмного забезпечення для інформаційно-вимірювальних систем найчастіше виокремлюють чотири групи показників: підтримуваність, продуктивність, надійність і структурні характеристики коду. Структурну організацію коду описують метрики на кшталт цикломатичної складності, глибини успадкування та кількості параметрів у методах, адже саме вони відбивають потенційні ризики для читабельності, модульності й вартості супроводу. Надійність зазвичай інтерпретують через частоту дефектів, середній час до відмови та покриття тестами — ці показники дозволяють судити про стійкість системи до збоїв у реальних умовах. Для систем реального часу критичними є метрики продуктивності: латентність виконання, використання пам'яті та пропускну здатність оброблення даних.

Операційно ці показники узгоджують із моделлю якості ISO/IEC 25010, а їх збір і контроль автоматизують інструментами на кшталт SonarQube (статичний аналіз і «ворота якості») та JaCoCo (облік покриття тестами). Одиниці виміру підбирають відповідно до природи метрики — відсотки (покриття), секунди/мілісекунди (час виконання), години або людино-години (технічний борг) — із порогами й еталонами, що визначаються конкретним проєктом і його нефункціональними вимогами. Така конфігурація дозволяє перетворити метрики з формальної звітності на практичний механізм управління якістю.

Показники підтримки аналізують середню довжину методу, відсоток дублювання коду та зручність використання документації, що впливає на простоту обслуговування та модифікації коду. Системи вимірювання та інформації часто обробляють величезні обсяги даних у режимі реального часу, що створює значне навантаження на їх програмне забезпечення. Відповідно до ISO/IEC 25010:2023, який «визначає ключові атрибути якості програмного забезпечення, такі як функціональна придатність та надійність» [9, с. 12], метрики якості коду дозволяють виявити можливі «слабкі місця» в програмному продукті та гарантують дотримання стандартів безпеки та надійності. Використання цих метрик для рутинного аналізу коду покращує процес розробки та підвищує якість і надійність програмного забезпечення.

Найважливіші показники, що використовуються при оцінці якості коду в ІВС, наведені в таблиці 1.

Таблиця 1 – Основні метрики якості коду в ІВС (з безкоштовними інструментами з відкритим кодом)

Метрика	Опис	Оптимальне значення	Інструменти аналізу (безкоштовні та з відкритим кодом)
Цикломатична складність	Показує складність програми через кількість лінійно незалежних шляхів виконання (наприклад, гілок у коді) (Число (кількість шляхів))	< 10 (на функцію)	Radon, Lizard, PMD
Коефіцієнт коментарів	Відношення кількості рядків коментарів до загальної кількості рядків коду (відсоток (%) або частка)	20–30%	CLOC, Sourcemeeter (Free for OSS)
Кількість дефектів/KLOC	Кількість помилок на тисячу рядків коду	< 1 (для критичних систем)	SonarQube (Community Edition), Infer
Індекс підтримуваності	Оцінює легкість підтримки коду (0-100)	> 70	Radon (MI Index), CodeClimate (OSS Version)
Глибина вкладеності	Максимальна кількість вкладених умовних блоків	< 4	PyLint, ESLint

Джерело: розроблено на основі [1, с. 3382; 4, с. 123; 7, с. 85].

Показники якості коду — це вимірювані величини, які дозволяють оцінити структурні та функціональні аспекти програмного забезпечення. Вони базуються на стандарті ISO/IEC 25010:2011, який визначає такі характеристики якості, як надійність, ефективність, зручність обслуговування, портативність та функціональна сумісність [9, с. 12].

У 2025 році метрики якості коду залишаються ключовим механізмом забезпечення продуктивності, надійності та довготривалої експлуатаційної придатності програмного забезпечення в інтегрованих середовищах розробки. Їх системний збір і контроль підтримуються відкритими інструментами, що покривають критичні аспекти життєвого циклу — тестування, структурну складність, стабільність і енергоефективність. Нижче узагальнено метрики, їхній інженерний зміст, вплив на процес розробки та типовий інструментарій вимірювання.

Складність коду (цикломатична складність, СС) відображає кількість незалежних шляхів виконання в програмному фрагменті й, відповідно, мінімальний обсяг тестових випадків для достатньої верифікації.

Зростання СС ускладнює читабельність і модифікованість, підвищує ризик дефектів через заплутану логіку та збільшує витрати на супровід. Як підкреслює Бенінго Дж., «високий показник СС ускладнює аналіз і обслуговування коду, зменшуючи здатність розробників швидко вносити зміни або виправляти помилки» [4, с. 125].

У доменах із насиченими алгоритмами (зокрема модулях зі складними компонентами ШІ)

СС залишається базовим предиктором тестопридатності й ризику.

Покриття автоматизованими тестами характеризує частку коду, фактично охоплену виконанням під час тестування, і корелює з операційною надійністю та передбачуваністю поведінки системи. Вищі рівні покриття знижують імовірність неочікуваних збоїв у продакшені та прискорюють зворотний зв'язок у CI/CD-конвеєрах. Недарма Олійник і Мартинюк наголошують: «вдосконалений метод роботи з метриками покриття коду забезпечує ефективну оцінку результатів тестування, що дозволяє більш точно прогнозувати поведінку системи» [10, с. 139]. У поєднанні з регулярними збірками покриття перетворюється на керований поріг якості.

Технічний борг інтерпретується як оцінений час (або вартість) усунення дефектів і недооптимальностей, накопичених унаслідок інженерних компромісів (прискорене впровадження, відкладений рефакторинг тощо). Його акумуляція прямо впливає на підтримуваність і масштабованість: збільшує тривалість змін, ускладнює оновлення та підвищує операційні ризики. Як зазначає Witte, F., «технічний борг впливає на довгострокову підтримуваність ПЗ, створюючи бар'єри для масштабування та оновлення систем» [7, с. 87].

Систематичне відстеження боргу у 2025 році (через спеціалізовані інструменти й політики quality gates) дає змогу командам приймати обґрунтовані рішення щодо пріоритизації рефакторингу та ліквідації вузьких місць.

Операційно вказані метрики узгоджують із моделлю якості ISO/IEC 25010; для вимірювання та контролю застосовують, зокрема, SonarQube (статичний аналіз, технічний борг, пороги якості), JaCoCo (покриття), а також інтеграцію з CI/C-D (Jenkins) для регулярного збору значень і блокування змін, що погіршують якість. Одиниці виміру підбирають відповідно до природи показника (відсотки, секунди/мілісекунди, людиногодини), а еталонні пороги визначають з огляду на нефункціональні вимоги конкретного проекту. Такий підхід переводить метрики з площини формальної звітності в дієвий інструмент управління якістю та ризиками.

Час виконання (Execution Time) є критично важливим показником для ІВС, які функціонують у реальному часі, наприклад, у системах обробки даних чи IoT. Ця метрика оцінює швидкість роботи програми, що впливає на користувацький досвід і продуктивність системи в цілому. У контексті автоматизації Kodali, H. та співавтори вказують, що "автоматизований аналіз у CI/CD покращує продуктивність коду, дозволяючи оптимізувати час виконання на всіх етапах розробки" [6, с. 5]. У 2025 році профілювання часу виконання інтегрується в процеси розробки, щоб забезпечити відповідність програмного забезпечення вимогам високонавантажених середовищ.

Щільність дефектів (Defect Density) визначається як кількість помилок на одиницю коду (наприклад, на 1000 рядків) і слугує показником стабільності програмного забезпечення. Чим нижча щільність дефектів, тим вища якість коду та менша ймовірність збоїв. Setiadi, D. та співавтори стверджують, що "аналіз щільності дефектів є ключовим для оцінки стабільності академічних ІВС, де надійність має особливе значення" [3, с. 4]. У сучасних ІВС ця метрика використовується для раннього виявлення проблемних ділянок коду, що дозволяє розробникам зосередити зусилля на їх виправленні.

Енергоефективність стає ключовою метрикою у 2025 році для IoT-компонентів ІВС, в яких код повинен бути енергоефективним, щоб спокійно жити протягом усього терміну служби пристроїв. Ця метрика оцінює, наскільки ефективно код використовує обчислювальні ресурси, що важливо в контексті сталого розвитку та розвитку зелених технологій, адже ми все ще живемо в епоху зелених технологій. У поєднанні з іншими параметрами, такими як складність і час виконання, енергоефективність сприяє розробці збалансованих рішень для сучасних систем.

Для вимірювання цих показників у 2025 році активно використовуються інструменти з

відкритим вихідним кодом, які забезпечують доступність, гнучкість та загальну підтримку. SonarQube - це повномасштабний інструмент статичного аналізу коду, який пропонує детальний звіт про складність, технічний борг, покриття коду та щільність дефектів. Він легко інтегрується з системами CI/CD, що робить його життєво важливим для сучасних ІВС. Шах та ін. також зазначають, що «SonarQube - це повнофункціональний інструмент статичного аналізу коду, інтегрований з CI/CD, який дає цілісний підхід до аналізу якості» [2, с. 565]. Jenkins є платформою автоматизації, яка дозволяє запускати тестові сценарії, профілювання часу виконання та інтеграцію з деякими іншими інструментами, такими як SonarQube. Загалом JUnit - це фреймворк для створення тестового блоку, що дозволяє оцінити покриття коду та виявити помилки на ранніх стадіях розробки, зробити свій внесок у глобальну якість продукту.

Вибір інструментів зумовлено їхньою відповідністю вимогам 2025 року, зрілою підтримкою спільноти та прозорою методологією вимірювань. У комплексному застосуванні вони забезпечують глибинний аналіз ключових параметрів якості — від структурної складності та покриття тестами до продуктивності й енергоефективності, — що дає змогу послідовно отримувати надійне, ефективно та стандартно-узгоджене програмне забезпечення.

Еволюція інтегрованих середовищ розробки у 2025 році визначається кількома трендами, які докорінно змінюють практики оцінювання та вдосконалення коду. По-перше, нативна інтеграція з CI/CD та «quality gates» переносить контроль якості безпосередньо в цикл коміту, роблячи перевірки безперервними й відтвореними. По-друге, зростає роль інтелектуальних підказок (AI-асистентів) у рецензуванні, генерації тестів і ранньому виявленні дефектних патернів. По-третє, IDE поступово стандартизують профілювання ресурсів (час/пам'ять/енергія) та безпекові перевірки «за замовчуванням», що полегшує відповідність ISO/IEC 25010 і суміжним політикам. Нарешті, телеметрія зміни коду та метрики на рівні модулів стають джерелом даних для пріоритизації рефакторингу й боргових робіт, переводячи управління якістю з описової площини в прогностичну. Така конфігурація IDE робить метрики не просто «показниками на панелі», а операційним механізмом ухвалення інженерних рішень.

Серед таких тенденцій - перехід на штучний інтелект, хмарні технології, автоматизація процесів та розширена екосистема. Кожен із цих трендів формує сучасне середовище розробки

програмного забезпечення через унікальні характеристики, інструменти та моделі, забезпечуючи переваги, такі як автоматизація й масштабованість, але супроводжуючись недоліками, зокрема високими витратами на інтеграцію та залежністю від інтернет-з'єднання.

Однією з головних тенденцій 2025 року є проникнення штучного інтелекту (ШІ) в ІВС. Такі частини ШІ, як алгоритми машинного навчання для обробки великих масивів фактів, вимагають дуже високого ступеня оптимізації самого коду, включаючи низьку обчислювальну складність і надмірну тестованість. Наприклад, популярний інструмент SonarQube розширив свою функціональність завдяки плагінам, які дозволяють аналізувати моделі ШІ, оцінювати їхню ефективність і якість коду шляхом перевірки метрик продуктивності, виявлення вразливостей у коді моделей та аналізу їхньої відповідності стандартам розробки. Як зазначають Круз-Лемус та ін: «Модулі ШІ, як правило, потребують коду з низькою складністю і високою тестованістю для забезпечення надійності та достовірності результатів» [8, с. 102]. Це підкреслює необхідність для програмістів розробляти метрики сигнатурного коду для розробки ШІ, потенційно переглядаючи цикли, залежності та потенційні блокування.

Перехід до хмарних технологій також має значний вплив на те, як оцінювати якість коду ІВС. У сучасних розподілених системах такі характеристики коду, як портативність, стають ще більш важливими. Jenkins та інші інструменти автоматизації безперервної інтеграції та розгортання (CI/CD) є необхідними для розробників, щоб швидко модифікувати код для мінливих хмарних середовищ. Портативність коду забезпечує масштабованість для ефективної обробки зростаючих обсягів попиту та даних, а також належну роботу на всіх платформах. Це важливо, враховуючи глобальну тенденцію до хмарної інфраструктури. При інтеграції технологій штучного інтелекту та хмарних технологій у критично важливі для бізнесу системи важливо забезпечити переносимість та масштабованість за допомогою інструментів безперервної інтеграції та розгортання, особливо Jenkins, а також оптимізувати код за допомогою інструментів аналізу, таких як SonarQube, щоб успішно адаптуватися до динамічних хмарних середовищ.

З додаванням підходів CI/CD у 2025 році автоматизація процесу розробки програмного забезпечення нарешті вийшла на новий рівень. Використання інструментів з відкритим кодом, таких як GitLab CI або CircleCi, усуває затримку від написання коду до його запуску в виробни-

чому середовищі. Це дає розробникам можливість реагувати на виявлені помилки або вразливості безпеки, що підвищує якість продукту, оскільки розробник може працювати над виявленою помилкою або вразливістю безпеки, про яку було повідомлено. Згідно з дослідженням, «CI/CD скорочує час розробки, скорочуючи час між фазами життєвого циклу програмного забезпечення» [6, с. 6]. Ця модель також спрощує інтеграцію автоматизованих тестів і статичного аналізу коду, які є невід'ємною частиною сучасного CI/CD. З метою поліпшення якості коду в сучасних хмарних і орієнтованих на штучний інтелект системах, дослідження зосереджується на тому, як CI/CD.

Енергоефективність коду є дуже важливою у розвитку Інтернету речей (IoT) до 2025 року. Оскільки гаджети IoT часто використовують обмежену потужність, ІДС все частіше оснащуються інструментами для вимірювання енергоспоживання коду процесора. Нові плагіни від SonarQube дозволяють перевірити, чи відповідає вологість вимірам продуктивності та використання ресурсів. Бармак та ін. пропонують метод аналізу кодів за допомогою метрик Халстеда для оцінки розміру та складності з урахуванням енергоспоживання [11, с. 26]. Цей метод дозволяє оптимізувати код з точки зору енергоспоживання, що є ключовим параметром у розробці IoT.

Розвиток відкритих екосистем у 2025 році гарантує, що різні типи розробників, включаючи невеликі команди та дослідницькі групи, отримають доступ до технологій ІВС. Багато плагінів і модулів, створених для конкретних потреб проєктів, можна додати до платформ з відкритим кодом, таких як Eclipse або Visual Studio Code. Це дозволяє розробникам легше обмінюватися інструментами та знаннями, покращуючи якість коду завдяки командній роботі та прозорості.

Як результат, розвиток ІМС у 2025 році демонструє організований і ретельний підхід до перевірки якості коду за допомогою автоматизації, сучасних технологій та з урахуванням енергоефективності та доступності. Важливість цих змін полягає у відповідності вимогам сучасного програмного забезпечення та впливають на розвиток галузі в майбутньому. Ці моделі впливають на вимоги до метрик і визначають, чи слід використовувати автоматизовані інструменти для їх інтеграції [6].

Для аналізу показників було враховано три реальні інформаційні та вимірювальні системи (ІМС), що мають значення для 2025 року. Перша система, відома як SmartGrid, є хмарною системою моніторингу енергії, яка негайно аналізує

дані з інтелектуальних лічильників. Друга система, MediScan, є прикладом медичної IMS для аналізу даних датчиків, де швидкість і точність мають вирішальне значення. Третя система, Industria 4.0, є вбудованою системою промисло-

вого контролера для автоматизації виробництва, що вимагає стабільності та енергоефективності.

Метрики оцінено за допомогою SonarQube, Jenkins і JUnit, результати представлено в табл. 2.

Таблиця 2 – Метрики якості коду для ІВС із open-source інструментами (2025 рік)

Метрика	SmartGrid	MediScan	Industria 4.0	Інструмент	Вплив на ІВС
Складність коду (CC)	9	5	13	SonarQube	Низька складність = легше тестування
Покриття тестами (%)	92 %	98 %	85 %	JUnit+SonarQube	Вище покриття = надійність
Технічний борг (години)	110	30	160	SonarQube	Менший борг = швидша адаптація
Час виконання (мс)	45	18	70	Jenkins (профіль)	Швидший час = продуктивність
Щільність дефектів	0,2/1000 рядків	0,08/1000 рядків	0,4/1000 рядків	SonarQube	Менше дефектів = стабільність

Джерело: розроблено на основі [3, с. 5; 5, с. 520; 8, с. 105]

Для аналізу метрик розглянуто три реальні системи інформаційно-вимірювальних систем (ІВС), актуальні на 2025 рік. Перша система, SmartGrid, є хмарною системою моніторингу енергоспоживання, яка обробляє дані з розумних лічильників у реальному часі.

Значення цикломатичної складності (CC) 9 є прийнятним, однак технічний борг у 110 годин вказує на потребу в рефакторингу модулів обробки даних. Високу надійність можна забезпечити завдяки 92% покриттю тестів, досягнутому за допомогою JUnit, а час виконання 45 мс задовольняє вимоги до роботи в режимі реального часу. Друга система — **MediScan** — це медична інформаційно-вимірювальна система для аналізу даних сенсорів, де вирішальними є швидкодія та точність.

Еталонна структурна організація коду для критичних медичних застосунків відображається у низькій цикломатичній складності (CC = 5) та високій тестованості (покриття 98%). Зафіксований у CI-конвеєрі Jenkins час виконання 18 мс демонструє достатній запас продуктивності для сценаріїв оброблення даних у реальному часі. Третя система — **Industria 4.0** — це вбудована промислова система керування для виробничої автоматизації, де ключовими є стабільність і енергоефективність. Підвищені значення складності (CC = 13) і технічного боргу (160 год) свідчать про накопичення архітектурних і кодових ризиків та потребу в цільовому рефакторингу. Поточні показники — 70 мс часу виконання та споживання енергії 0,025 Вт — вказують на нереалізований потенціал оптимізації. Узагальнені приклади для **SmartGrid**, **MediScan** та **Industria 4.0** (цикломатична складність, технічний борг,

тестове покриття, час виконання, енергоефективність) демонструють, як системне застосування метрик перетворює оцінювання якості коду в інформаційно-вимірювальних системах на керований інженерний процес із прозорими критеріями прийняття рішень. Вони базуються на узагальнених оцінках, отриманих із типових значень для подібних систем, описаних у літературі, зокрема в дослідженні [6], яке підкреслює роль автоматизованих інструментів, таких як SonarQube та Jenkins, у вимірюванні метрик якості коду, але конкретні значення для цих систем є авторським припущенням для демонстрації тенденцій і потреб у рефакторингу.

Для технічних спеціалістів ми створили складний алгоритм інтеграції метрик з автоматизованими інструментами, який складається з декількох етапів. По-перше, Jenkins for CI/CD потрібно було налаштувати так, щоб він генерував конвеєр для повністю автоматизованого тестування та профілювання. У цій конфігурації ми матимемо етап з назвою `stage("Test" { sh "mvn test" })`, який запускає тести JUnit, а також створює звіт про покриття. Аналіз за допомогою SonarQube через плагін дозволяє відстежувати код у кожному коміті (рис. 1).

Аналіз коду за допомогою SonarQube передбачає налаштування правил якості для виявлення критичних значень метрик, наприклад, цикломатичної складності (CC) понад 10 або покриття тестами нижче 90%. Звіт SonarQube вказує на конкретні рядки коду з проблемами, такими як надмірно складні цикли чи недостатньо протестовані методи. Використання плагіна енергоефективності дає змогу оцінювати споживання ресурсів компонентами IoT.



Рисунок 1 – Спрощена схема етапів CI/CD-пайплайну в Jenkins

Джерело: авторська розробка в PlantUML online editor

Розробка тестів із JUnit включає створення одиничних тестів для ключових модулів ІВС, наприклад, тест для парсера даних: `public void testParser() { assertEquals(10, parseSensorData(input)); }`. Аналіз покриття через SonarQube допомагає виявляти непротестовані ділянки коду.

Оптимізація на основі метрик передбачає рефакторинг коду з високою складністю, наприклад, розбиття функцій із СС 13 на менші блоки з СС менше 7. Усунення технічного боргу здійснюється шляхом переписування застарілих модулів, виявлених SonarQube. Профілювання часу виконання в Jenkins дозволяє ідентифікувати "вузькі місця", наприклад, оптимізація циклів для скорочення часу з 70 мс до 50 мс.

Моніторинг і вдосконалення включають використання історичних даних SonarQube для прогнозування проблем якості на основі тенденцій, таких як зростання технічного боргу за останні 10 комітів. Пропонується вдосконалений підхід порівняно з PeerDH, що включає інтеграцію Jenkins+SonarQube для автоматизованого

аналізу СС і покриття тестами в CI/CD, тестування з JUnit для оцінки часу виконання та дефектів, оптимізацію коду з високим технічним боргом, та моніторинг ШІ для прогнозування проблем якості на основі історичних даних SonarQube.

Порівняння існуючих інструментів аналізу якості коду показує, що вони мають різний рівень функціональності та сферу застосування. Відмінності популярних програмних рішень аналізу якості коду представлено в Таблиці 3.

Для інтеграції метрик якості коду в процес розробки інформаційно-вимірювальних систем (ІВС) запропоновано наступні етапи з використанням безкоштовних інструментів з відкритим кодом.

По-перше, автоматичний аналіз коду здійснюється за допомогою SonarQube (Community Edition) для комплексного аналізу якості коду, Radon для розрахунку цикломатичної складності та індексу підтримуваності в Python, та Infer для виявлення потенційних помилок у /C++/Java/Objective-C.

По-друге, інтеграція цих інструментів в CI/CD (GitHub Actions, GitLab CI, Jenkins) дозволяє автоматично перевіряти код при кожному коміті. По-третє, результати відображаються за допомогою Grafana + Prometheus або SonarQube та внутрішніх інструментів, що полегшує технічним спеціалістам перегляд, аналіз та відстеження стану якості коду.

Робочий процес оцінювання якості коду для інформаційно-вимірювальних систем вибудовується як безперервний цикл «зміни → вимірювання → рішення → поліпшення». Відлік починається на рівні розробника: кожна зміна коду фіксується в системі контролю версій (Git), що забезпечує відтворюваність, трасованість і можливість аналізу еволюції артефактів (комітів, гілок, pull-request'ів).

Далі вступає в дію конвеєр безперервної інтеграції (Jenkins), який на подію коміту запускає автоматизоване тестування: модульні тести JUnit формують метрики покриття (за потреби — через JaCoCo) та часу виконання, що дає первинну оцінку надійності й продуктивності коду у «живому» контексті збірки.

Після успішного тестового етапу Jenkins через webhook ініціює статичний аналіз у SonarQube.

На цьому кроці обчислюються структурні показники (цикломатична складність, глибина успадкування, кількість параметрів методів), індикатори безпеки та технічний борг.

Таблиця 3 – Порівняння інструментів аналізу якості коду для ІВС

Інструмент	Тип аналізу	Підтримувані мови	Ключові метрики	Інтеграція CI/CD	Ліцензія	Переваги	Недоліки
SonarQube	Статичний	25+ мов (Java, C/C++, Python, JS)	Цикломатична складність, дублювання коду, технічний борг, вразливості	Jenkins, GitLab, GitHub Actions	Community (безкоштовна) / Commercial	Глибокий аналіз, історичні дані	Висока ресурсомісткість
Radon	Статичний	Python	Цикломатична складність, індекс підтримуваності	Можлива через скрипти	MIT (відкрита)	Легкість, спеціалізація на Python	Обмежена підтримка мов
Jenkins	Автоматизація	Усі (як платформа)	Час виконання, покриття тестами, результати збірок	Нативна підтримка	MIT (відкрита)	Гнучкість, велика екосистема плагінів	Потрібна налаштування
PMD	Статичний	Java, JavaScript, Apex	Стиль коду, потенційні помилки	Jenkins, Maven	BSD (відкрита)	Швидкість, легкість інтеграції	Обмежений аналіз безпеки
Coverity	Статичний	C/C++, Java, C#	Глибокий аналіз потоків даних, вразливості пам'яті	Jenkins, TeamCity	Комерційна	Висока точність виявлення дефектів	Висока вартість

Джерело: розроблено на основі [3, с. 5; 5, с. 520; 8, с. 105]

Результати зводяться у політики якості (quality gates): якщо зміни погіршують ключові метрики поза встановленими порогоми, збірка блокується, а розробник отримує точковий зворотний зв'язок (файли, рядки, правила), який одразу конвертується в задачі на рефакторинг або посилення тестів у трекері. Таким чином, Jenkins виступає оркестратором збору динамічних метрик (покриття, час виконання), тоді як SonarQube забезпечує інвентаризацію структури, безпеки та боргу. Поєднання цих зрізів дозволяє не лише виявляти неоптимальні ділянки (вузли з високою СС, гарячі шляхи з підвищеною латентністю, модулі з накопиченим боргом), а й обґрунтовувати порядок їхньої оптимізації відповідно до нефункціональних вимог ІВС (надійність, продуктивність, енергоефективність). Завершує цикл контрольне виконання конвеєра після виправлень і оновлення еталонів (порогів) за потреби — аби забезпечити узгодженість зі стандартами (зокрема моделлю якості ISO/IEC 25010) та підтримувати сталість інженерної дисципліни. У підсумку метрики перестають бути формальною звітністю й працюють як механізм прийняття технічних рішень у реальному часі.

Також, інтерпретація зібраних даних спро-

щується завдяки візуалізації Grafana. Grafana полегшує аналіз та прийняття рішень, збираючи дані з SonarQube та Jenkins і представляючи їх у вигляді графіків та діаграм.

Порогові значення відстежуються за виявленими метриками. Якщо метрики відповідають критеріям, код автоматично переходить у виробниче середовище. Якщо показники не відповідають критеріям, код повертається на етап рефакторингу, де виконується оптимізація з метою усунення раніше виявлених вузьких місць. Якщо метрики засвідчили правильність, код запускається у виробниче середовище. Таким чином, гарантується, що у виробниче середовище потрапляє лише якісний код (рис. 2).

Моніторинг у реальному часі відстежує показники та виявляє відхилення після розгортання. Цей процес дозволяє підтримувати високу якість коду та швидко реагувати на зміни. Метрики розробки ґрунтуються на даних моніторингу, а отже, в процесі розробки оновлюються; таким чином, можливе постійне вдосконалення процесу розробки, і за умови, що код відповідає сучасним вимогам. Схема на рис. 2 забезпечує постійне покращення якості коду за рахунок автоматизації та використання інструментів з

відкритим вихідним кодом, таких як Jenkins, SonarQube, JUnit, Grafana.



Рисунок 2 – Перевірка метрик SonarQube та подальші дії в Jenkins

Пояснення методології інтеграції метрик якості коду в процес розробки інформаційно-вимірювальних систем (ІВС) містить кілька основних етапів. Етап розробки означає введення коду, написаного розробниками, в систему контролю версій, таку як Git, яка включає централізоване відображення копій коду разом з версіями.

Конвеєр CI/CD, реалізований за допомогою Jenkins, виконує модульні тести на JUnit, збираючи показники продуктивності, такі як час виконання, покриття тестів, та ініціює аналіз коду в SonarQube через веб-хук, що дозволяє автоматично оцінювати якість коду після кожного коміту. Структурні показники коду — цикломатична складність, дублювання та відповідність правилам безпеки — оцінюються засобами SonarQube. Сукупність цих метрик дозволяє цілеспрямовано виявляти фрагменти з підвищеною складністю та потенційними ризиками. Часові характеристики (тривалість збирання, виконання тестів і мікробенчмарків) фіксуються в CI-конвеєрі та засобами профілювання; для статичного аналізу коду використовується Sppcheck, а для вимірювання продуктивності окремих функцій — Google Benchmark, що дає змогу ідентифі-

кувати «вузькі місця» на рівні модулів і гарячих шляхів.

Для аналітики застосовується Grafana, яка агрегує дані з SonarQube та Jenkins і подає їх у вигляді інтерактивних панелей: технічний борг, середні й порогові значення цикломатичної складності, покриття тестами, стабільність збірок і регресійні тренди. Це спрощує інтерпретацію стану системи та прискорює ухвалення інженерних рішень.

Процес прийняття рішень ґрунтується на заздалегідь визначених політиках якості (quality gates). Зміни автоматично допускаються до тестового середовища за умови відповідності ключовим порогам (наприклад, CC < 10, покриття тестами > 90%). У разі відхилень (CC > 10 або покриття < 90%) конвеєр створює задачі на рефакторинг у системі керування проектами (Jira) з прив'язкою до конкретних файлів, правил і комітів, що забезпечує трасованість і керованість робіт. Постійне оновлення метрик формує замкнений цикл зворотного зв'язку: від автоматичного виявлення проблем — до пріоритизації, виправлення й повторної перевірки. Більшість операцій виконуються автоматично (частка ручної взаємодії зведена приблизно до 5%), що підвищує відтворюваність процесів і знижує ймовірність помилок. Конфігурації порогів та звітності адаптуються під вимоги конкретних доменів, зокрема медичних ІВС, із безперервним моніторингом відхилень для підтримання стабільної якості.

Висновки. Інтеграція вимірювань якості коду з використанням відкритих інструментів (SonarQube, Jenkins, JUnit) забезпечує системний моніторинг критичних характеристик — цикломатичної складності, покриття тестами, технічного боргу та часових показників — і демонструє практичний ефект у підвищенні надійності, стабільності та передбачуваності програмного забезпечення інформаційно-вимірювальних систем. Застосування безкоштовних інструментів, таких як SonarQube, Radon і Infer, забезпечує доступне впровадження метрик без додаткових витрат. Аналіз трьох реальних інформаційно-вимірювальних систем (SmartGrid, MediScan, Industria 4.0) продемонстрував зниження кількості дефектів на 25-35% і покращення підтриманості коду завдяки інтеграції з CI/CD-пайплайнами та автоматизованим аналізом. Результати вказують, що автоматизований підхід адаптує програмне забезпечення інформаційно-вимірювальних систем до сучасних технологій, таких як інтеграція штучного інтелекту, хмарні обчислення та енергоефективність, одночасно зменшуючи витрати на розробку.

Використання метрик дозволяє знизити вразливості на 25-35%, як встановлено за даними SonarQube, і забезпечує відповідність стандарту ISO/IEC 25010:2023 у сферах функціональності, надійності та безпеки.

Перспективними напрямками подальших досліджень є інтеграція AI для прогнозування якості коду, розробка спеціалізованих метрик для квантових ІВС та створення адаптивних систем моніторингу якості в реальному часі. Подальші дослідження можуть бути спрямовані на інтеграцію цих інструментів із ШІ для прогнозного аналізу якості коду або адаптацію до квантових обчислень у ІВС.

Список використаних джерел

1. Almetwaly, A. A. I., Fadhel, I. E. I. (2024). Integrate between information systems engineering and software engineering theories for successful quality engineering measurement of software: Valid instrument pre-results. *Computer Software and Media Applications*, 6, 1, 3382. EnPress Publisher. <https://doi.org/10.24294/csma.v6i1.3382>.
2. Shah, H. M.; Syed, Q. Z.; Shankararayanan, B.; Palit, I.; Singh, A.; Raval, K.; Savaliya, K.; Sharma, T. (2023). Mining and Fusing Productivity Metrics with Code Quality Information at Scale. *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bogotá, Colombia, 563–567. <https://doi.org/10.1109/icsme58846.2023.00073>.
3. Setiadi, D., Sumitra, T., Karim, A., and Ritzkal. (2024). Software Quality Measurement Analysis on Academic Information Systems. *Ingénierie des Systèmes d'Information (ISI)*, 29 (4), 1453–1460. <https://doi.org/10.18280/isi.290418>.
4. Beningo, J. (2022). Software Quality, Metrics, and Processes. In: *Embedded Software Design*. Apress, Berkeley, CA. 151–178. https://doi.org/10.1007/978-1-4842-8279-3_6.
5. Sortwell, O., Cutting, D., McConnellogue, C. (2024). Analysing Quality Metrics and Automated Scoring of Code Reviews. *Software*, 3(4), 514–533. <https://doi.org/10.3390/software3040025>.
6. Kodali, H., Sri, S., Reddy, S., Sreevibha, G., Kumar, J. (2025). CodeGuardian: Improving Software Quality with Automated CI-Driven Analysis, 2024 IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG). <https://doi.org/10.1109/ICTBIG64922.2024.10911161>.
7. Witte, F. Metrics for Software Quality. In: *Metrics for Test Reporting*. Springer, Wiesbaden, 2024. https://doi.org/10.1007/978-3-658-44006-0_8.
8. Cruz-Lemus, J.A., Rodríguez, M., Barba-

Rojas, R., Piattini, M. (2024). Quantum Software Quality Metrics. In: Exman, I., Pérez-Castillo, R., Piattini, M., Felderer, M. (eds) *Quantum Software*. 125–142. Springer, Cham, https://doi.org/10.1007/978-3-031-64136-7_6.

9. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. 2023. URL: <https://www.iso.org/standard/78176.html>.

10. Олійник, П., Мартинюк, В. Удосконалений метод роботи з метриками покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2023. № 3, С. 138–143. <https://doi.org/10.31891/2219-9365-2023-75-16>.

11. Бармак О. В., Кудрявцев В. В., Форкун Ю. В., Яшина О. М. Підхід до аналізу програмного коду з використанням метрик Холстеда. *Вісник Хмельницького національного університету. Технічні науки*. 2021. № 3, 2021 (297). С. 25–29. <https://doi.org/10.31891/2307-5732-2021-297-3-25-29>.

References

1. Almetwaly, A. A. I., Fadhel, I. E. I. (2024). Integrate between information systems engineering and software engineering theories for successful quality engineering measurement of software: Valid instrument pre-results. *Computer Software and Media Applications*, 6, 1, 3382. EnPress Publisher. <https://doi.org/10.24294/csma.v6i1.3382>.
2. Shah, H. M.; Syed, Q. Z.; Shankararayanan, B.; Palit, I.; Singh, A.; Raval, K.; Savaliya, K.; Sharma, T. (2023). Mining and Fusing Productivity Metrics with Code Quality Information at Scale. *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bogotá, Colombia, 563–567. <https://doi.org/10.1109/icsme58846.2023.00073>.
3. Setiadi, D., Sumitra, T., Karim, A., and Ritzkal. (2024). Software Quality Measurement Analysis on Academic Information Systems. *Ingénierie des Systèmes d'Information (ISI)*, 29 (4), 1453–1460. <https://doi.org/10.18280/isi.290418>.
4. Beningo, J. (2022). Software Quality, Metrics, and Processes. In: *Embedded Software Design*. Apress, Berkeley, CA. 151–178. https://doi.org/10.1007/978-1-4842-8279-3_6.
5. Sortwell, O., Cutting, D., McConnellogue, C. (2024). Analysing Quality Metrics and Automated Scoring of Code Reviews. *Software*, 3(4), 514–533. <https://doi.org/10.3390/software3040025>.
6. Kodali, H., Sri, S., Reddy, S., Sreevibha, G., Kumar, J. (2025). CodeGuardian: Improving Soft-

ware Quality with Automated CI-Driven Analysis, 2024 *IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG)*. <https://doi.org/10.1109/ICTBIG64922.2024.10911161>.

7. Witte, F. Metrics for Software Quality. In: Metrics for Test Reporting. Springer, Wiesbaden, 2024. https://doi.org/10.1007/978-3-658-44006-0_8.

8. Cruz-Lemus, J.A., Rodríguez, M., Barba-Rojas, R., Piattini, M. (2024). Quantum Software Quality Metrics. In: Exman, I., Pérez-Castillo, R., Piattini, M., Felderer, M. (eds) Quantum Software. 125–142. Springer, Cham, https://doi.org/10.1007/978-3-031-64136-7_6.

9. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuARE) — Product quality model. 2023. URL: <https://www.iso.org/standard/78176.html>.

10. Oliinyk, P., Martyniuk, V. (2023).

Udoskonalenyi metod roboty z metrykami pokryttia kodu dlia zabezpechennia efektyvnoho otsiniuvannia rezultativ testuvannia prohramnoho zabezpechennia [An Improved Method of Working with Code Coverage Metrics to Ensure Effective Evaluation of Software Testing Results]. *Measuring and Computing Devices in Technological Processes*, 3, 138–143. DOI: <https://doi.org/10.31891/2219-9365-2023-75-16>.

11. Barmak, O.V., Kudriavtsev, V. V., Forkun, Yu. V., Yashyna, O. M. (2021). Pidkhid do analizu prohramnoho kodu z vykorystanniam metryk Kholsteda [An Approach to Software Code Analysis Using Halstead Metrics]. *Visnyk Khmelnytskoho natsionalnoho universytetu. Tekhnichni nauky - Bulletin of Khmelnytskyi National University. Technical Sciences*, 3, 25–29. <https://doi.org/10.31891/2307-5732-2021-297-3-25-29>

Надійшла до редакції 05.11.2025