DOI 10.31891/2307-5732-2025-347-28
УДК 004.4

КУСТОВСЬКИЙ РОМАН
Хмельницький національний університет
https://orcid.org/0000-0003-0785-7202
e-mail: roma.kust99@gmail.com
ЯШИНА ОКСАНА
Хмельницький національний університет
ORCID ID: https://orcid.org/0000-0001-7816-1662
e-mail: oksana.yashyna@ukr.net

МЕТОДИ УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В роботі представлені методи управління якістю програмного з-абезпечення. Під якістю програмного забезпечення розуміють те, наскільки програмний продукт відповідає вимогам та потребам своїх користувачів. Мова йде як про саме програмне забезпечення, так і процеси його створення. Якісно зроблене програмне забезпечення характеризується достатньою вільністю від помилок та дефектів, вчасною в рамках бюджету поставкою, відповідністю вимогам та придатністю для обслуговування. Структуроване управління якістю програмного забезпечення - це його забезпечення (SQA), планування (SQP) та контроль (SQC). шляхом до створення високоякісного програмного забезпечення є впровадження ефективного управління якістю з відповідними методами. Після наведених вище етапів тестування є основним видом діяльності, що виявляє та вирішує технічні проблеми у вихідному коді програмного забезпечення, оцінює загальну зручність використання, продуктивність, безпеку та сумісність продукту.

Серед методів вдосконалення процесу тестування програмного забезпечення та підвищення якості програмних продуктів можна виділити: планування процесу тестування та контролю якості, використання орієнтованого на тестування управління розробкою програмного забезпечення, проведення офіційних технічних оглядів, забезпечення відповідного робочого середовища для команди, впровадження приймального тестування користувача (UAT), оптимізація використання автоматизованих тестів, впровадження дослідницького та спеціального тестування, використання вимірювання якості коду, ефективні звіти про помилки, отримання максимальної віддачі від інструментів управління тестуванням. Запропоновані методи можуть бути застосовані для моніторингу та управління якістю програмного забезпечення.

Ключові слова: якість програмного забезпечення, метод, тестування, контроль якості, розробка програмного забезпечення, автоматизавані тести.

KUSTOVSKYI ROMAN
YASHYNA OKSANA
Khmelnytskyi National University

METHODS OF SOFTWARE QUALITY MANAGEMENT

The paper presents methods of software quality management. Software quality is defined as the extent to which a software product meets the requirements and needs of its users. It refers to both the software itself and the processes of its creation. Well-made software is characterised by being sufficiently free from errors and defects, delivered on time and on budget, and meeting requirements and being maintainable. Structured software quality management is about software quality assurance (SQA), planning (SQP) and control (SQC). the way to create high-quality software is to implement effective quality management with an appropriate methodology. After the above steps, testing is the main activity that identifies and resolves technical problems in the software source code, evaluates the overall usability, performance, security and compatibility of the product.

Software quality assurance is its prevention, which involves the creation of instructions, implementation of standards, and improvement of processes throughout the software life cycle to ensure that the quality criteria are met. Planning, in turn, deals with setting standards, instructions, ensuring their inclusion in project planning by creating a quality plan, which often contains a description of quality control activities with the definition of time, performers, methods and resources.

Among the methods of improving the software testing process and improving the quality of software products are: planning the testing and quality control process, using test-oriented software development management, conducting formal technical reviews, providing an appropriate working environment for the team, implementing user acceptance testing (UAT), optimising the use of automated tests, implementing exploratory and ad hoc testing, using measurement, and so on. The proposed methods can be applied to software quality monitoring and management. The presented methodology suggests that for effective software quality management it is necessary to cover all its key aspects, such as effective planning, a test-oriented approach to quality management and a specially formed professional team.

Keywords: software quality, method, testing, quality control, software development, automated tests.

Вступ та постановка проблеми

Сучасний цифровий світ змушує бізнес – сфери покладатися на програмне забезпечення, яке відповідно має працювати належним чином. В цьому аспекті є важливим управління якістю програмного забезпечення (SQM), що являє собою безперервний процес гарантуваня виконання належним чином програмним забезпеченням своєї функції. Нехтування якістю програмного забезпечення може призвести до невиправданих витрат та значних ризиків, як от наприклад у 2023 році сталися збої в обслуговуванні United Airlines внаслідок невдалого оновлення програмного забезпечення чи було вимкнено ChatGPT через помилку в бібліотеці з відкритим кодом, що давала доступ деяким користувачам до перегляду заголовків з історії чату інших активних користувачів.

Зрозуміло, що для компаній важливо швидко виводити продукт з новими функціями на ринок послуг, при цьому необхідно водночас критично відноситись до побудови самого програмного забезпечення. Саме управління якістю програмного забезпечення дозволяє впевнено уникати ризиків,

дотримуватися усіх вимог, зменшувати свої витрати, приймати правильні рішення на основі отриманих даних, маючи успіх в надзвичайно конкурентній цифровій економіці. Якість програмного забезпечення є неодмінною умовою кожного успішного бізнесу, що займається його розробкою, адже це сприяє позитивній репутації на ринку та й подальшій продуктивності працівників, які пишаються результатами своєї праці. До того ж якісний кінцевий продукт задовільняє клієнта, який в майбутньому знову може звернутися по послугу і дати позитивні рекомендації, що вкрай важливо для іміджу. Діючи на випередження, можна значно зменшити витрати, зусилля і час, необхідні для корегування неякісного кінцевого продукту, адже цей процес є довготривалим, недешевим і складним.

Аналіз останніх досліджень

У дослідженні [1] науковці узагальнили існуючі термінологічні трактування поняття «якість програмного забезпечення», на основі чого зроблено висновки щодо відповідності термінів для оцінки якості загального програмного забезпечення до процесу оцінки якості програмного забезпечення інтелектуальних систем прийняття рішень. Обгрунтовано також, що якість програмного забезпечення інтелектуальних систем прийняття рішень є комплексним багатокритеріальним показником, який бере до уваги не тільки якість роботи окремого програмного модуля як підсистеми, а й причинно-наслідкові зв'язки елементів самої програмної системи. Описано основні розбіжності в оцінці якості програмного забезпечення при застосуванні функціонального та формального підходів.

В роботі [2] розглядають якість програмного забезпечення як комплексне поняття. Оскільки стандарти виділяють якість процесу розробки, внутрішню і зовнішню якість програмного продукту, якість програмного продукту на стадії використання, то для кожного з цих компонентів є набір метрик для визначення якості програмного продукту, що називають моделлю якості програмного забезпечення. Методи і алгоритми автоматичного розрахунку метрик складності програмного забезпечення за допомогою програмних засобів служать отриманню комплексного формального звіту про якість програмного забезпечення за нетривалий час, що уможливлює об'єктивний моніторинг рівня якості протягом всього життєвого циклу.

В наукових пошуках [3] розроблено систему управління якістю програмного забезпечення для визначення стану його якості на кожному ієрархічному рівні системи з урахуванням вартості розроблення. Це метод відбору напружених варіантів стану системи якості програмного забезпечення за вхідними критерями чи агрегованими показниками, який уможливлює визначення поточного стану системи управління якістю програмного забезпечення. А також метод вибору оптимального варіанту системи управління якістю програмного забезпечення з множини допустимих альтернатив з урахуванням структури критеріїв і агрегованих показників на кожному ієрархічному рівні системи. Задача вибору враховує такі критерії, як портативність продукту та зручність його супроводу, безпека продукту та його сумісність, надійність роботи продукту та зручності його використання, функціональна придатність продукту та ефективність виконання, що належать до задач багатокритеріальної оптимізації.

Дослідники [4] описують удосконалення статичних моделей надійності програмного забезпечення за допомогою методів машинного навчання для вибору метрик коду, що найсильніше впливають на його надійність. У дослідженні використано злитий датасет з репозиторію PROMISE Software Engineering, що має дані про тестування програмних модулів п'яти програм (KC1, KC2, PC1, CM1, JM1) та двадцять одну метрику коду. Для вибірки зроблено вибір найважливіших ознак, що мають вплив на якість програмного коду за допомогою таких методів вибору ознак як Boruta, Step-wise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python.

У дослідженні [5] проведено тестування продуктивності, спрямоване на забезпечення експлуатаційної ефективності програмних систем та проаналізували тестові набори продуктивності систем із відкритим вихідним кодом для вивчення величини їхнього покриття коду разом з часом їхнього виконання. Результатом є те, що тести продуктивності досягають значно нижчого покриття коду, ніж функціональні тести, що підкреслює значний компроміс між покриттям і часом виконання. Можна припустити для перспективних досліджень, що автоматизовані методи генерації тестів можуть не гарантувати доступне тестування продуктивності через пов'язані з цим часові витрати, що ставить нові завдання щодо генерації тестів продуктивності.

Загальний аналіз показує, що методологічна основа управління якістю програмного забезпечення є об'ємним та багагатограним процесом, оскільки включає і технічні характеристики, і управлінський потенціал, і людський ресурс. Лише правильне поєднання цих факторів може створити позитивні передумови для ефективного управління якістю програмного забезпечення.

Метою роботи є визначення методології управління якістю програмного забезпечення, що передбачає формулювання методів вдосконалення процесу тестування програмного забезпечення та підвищення якості програмних продуктів Це дозволить підвищити точність і оперативність тестування програмного забезпечення на всіх етапах його життєвого циклу.

Основна частина

Під якістю програмного забезпечення розуміють те, наскільки програмний продукт відповідає вимогам та потребам своїх користувачів. Мова йде як про саме програмне забезпечення, так і процеси

його створення. Якісно зроблене програмне забезпечення характеризується достатньою вільністю від помилок та дефектів, вчасною в рамках бюджету поставкою, відповідністю вимогам та придатністю для обслуговування. Ще Стандарт ISO 8402-1986 сформулював якість як «сукупність характеристик і властивостей продукту або послуги, які мають здатність відповідати наявним потребам» [6]. В своєму оригінальному вигляді якість важко описується, але легко пізнається за її присутності. З точки зору виробництва якість має відповідати нормативним стандартам.

Придатністю до задоволення його потреб є основним аспектом якості для користувача, що може включати в себе і гарний дизайн, і тривалість в експлуатації, і надійність, і хорошу функціональність, і адекватне співвідношення ціни та якості. Якість програмного забезпечення варто розглядати за трьома вимірами: функціональність, що передбачає, наскільки добре виконується те, для чого було призначене, структурність самого коду, ефективність безпосередньо процесу розробки.

Як і будь-яким явищем, якістю програмного забезпечення можна і треба управляти. SQM являє собою виконання того, щоб якість програмного забезпечення підтримувалась впродовж усього життєвого циклу його розробки. SQM сприяє розробці програмного забезпечення, що відповідає нормативним стандартам, вимогам високої якості та бізнес-цілям, в цілому належно працює, забезпечуючи взаємодію з користувачем. Управління якістю програмного забезпечення включає нагляд та вдосконалення всіх аспектів розробки, розгортання та обслуговування програмного забезпечення для задоволення потреб користувача.

Якщо ж говорити про більш структуроване управління якістю програмного забезпечення, то це його забезпечення (SQA), планування (SQP) та контроль (SQC). Забезпечення якості програмного забезпечення іншими словами є його профілактикою, що передбачає створення інструкцій, запровадження стандартів, вдосконалення процесів протягом всього життєвого циклу програмного забезпечення для гарантування відповідним критеріям якості. Планування в свою чергу займається встановленням стандартів, інструкцій, забезпечуючи їх включення в планування проекту створенням плану якості, який часто містить опис діяльності з контролю якості з визначенням часу, виконавців, методів та ресурсів.

Кінцевою метою контролю якості є запобігання дефектам, а не їх виявлення шляхом тестування для заощадження часу і коштів. Добре розроблений процес контролю передбачає, що кожен учасник розуміє усю відповідальність та вимоги до своєї роботи. План якості може допомогти завчасно виявити і вирішити потенційні проблеми з якістю ще до того, як вони вплинуть на кінцевий продукт, та переконатися у відповідності програмного забезпечення необхідним стандартам на кожному етапі його життєвого циклу. SQC має реактивний характер, дотримується плану якості для гарантування відповідності нормативним стандартам якості та, будучи елементом управління якістю програмного забезпечення, перевіряє, чи відповідає програмний продукт вимогам, які встановлені під час процесу SQA із дотриманням SQP. Остаточною метою SQC є перевірка коду ще до ретельного тестування з метою знаходження та виправлення помилок перед запуском у виробництво програмного забезпечення.

Таким чином, шляхом до створення високоякісного програмного забезпечення є впровадження ефективного менеджменту якості з відповідною методологією. Після наведених вище етапів тестування є основним видом діяльності, що виявляє та вирішує технічні проблеми у вихідному коді програмного забезпечення, оцінює загальну зручність використання, продуктивність, безпеку та сумісність продукту. Тестування водночас виступає і основною частиною забезпечення якості і невід'ємною частиною процесу розробки програмного забезпечення. Якщо проект невеликий, то створюється стратегія тестування як частина плану тестування, для більшого ж керівник проекту має розробити стратегію тестування у формі окремого статистичного документу, на основі якого можна далі розробляти кожен план тестування. План тестування є документом, який містить інформацію про те, що, коли і як тестувати, хто це робитиме, описує обсяг, заходи, цілі тестування, що допомагає контролювати ризики. В цьому документі має міститися розклад усіх необхідних заходів тестування для контролю часу роботи команди, ролі її членів для чіткого розуміння дій. Не існує універсального способу створення плану тестування, так як це залежить від процесів, нормативних стандартів, інструментів керування тестуванням, заведених у відповідній компанії.

Підготовка ефективних тестів це необхідна частина вдосконалення тестування програмного забезпечення. Міжнародна кваліфікаційна рада з тестування програмного забезпечення ISTQB, яка є світовим лідером із сертифікації компетенцій у тестуванні програмного забезпечення, визначає, що «тестовий приклад – це набір вхідних значень, це передумова виконання, очікування результатів і постумов виконання, розроблених для конкретної цілі чи умови тестування, наприклад, для виконання конкретного програмного шляху або для перевірки відповідності певній вимозі» [7]. Для тестерів це є один із ключових інструментів.

Серед методів вдосконалення процесу тестування програмного забезпечення та підвищення якості програмних продуктів можна виділити:

1. Планування процесу тестування та контролю якості.

Сам план тестування рекомендується робити коротким, інформативним, без повторів і неточностей, проте з необхідними деталями, постійно оновлювати, ділитися ним з усіма зацікавленими

сторонами. Правильно сформований план тестування буде чітко відображати якість тестування командою.

2. Використання орієнтованого на тестування управління розробкою програмного забезпечення.

Гарним способом підвищення якості програмного забезпечення є впровадження тестово-орієнтованих підходів до управління. Серед способів досягнення цього виокремлюють використання екстремального програмування (XP) – методології розробки програмного забезпечення спрямованого на створення програмного забезпечення вищої якості з можливістю адаптації до мінливих вимог. Існує дві найпоширеніші практики XP, пов'язані з тестуванням. Керована тестуванням розробка (TDD), у якій тести створюються до впровадження коду. Згідно з цим перед кожною новою функцією пишеться розробником автоматизований тестовий приклад. Спочатку цей тест буде невдалий, а вже наступним етапом є написання коду, який зосереджений на функціональності, що подолати тест. Після цих етапів розробник рефакторює код для проходження всіх тестів.

Іншою практикою екстремального програмування є парне програмування, що передбачає одночасну роботу за одним комп'ютером двох інженерів. Один пише код, другий – спостерігає за процесом, вносячи за необхідності необхідні пропозиції. Цей метод створює високу якість коду, оскільки помилки коригуються вже по ходу його написання.

3. Проведення офіційних технічних оглядів.

Формальний технічний огляд (FTR) являє собою діяльність інженерів програмного забезпечення для виявлення функціональних і логічних помилок на ранніх стадіях. FTR є груповою зустріччю, під час якої відповідний персонал перевіряє розроблене програмне забезпечення на відповідність визначеним вимогам і стандартам.

Основною метою рецензійної зустрічі є представлення продукту решті рецензентів, в результаті чого всі учасники процесу повинні прийняти продукт або ж запропонувати зміни, обговоривши часові рамки.

Покрокове керівництво передбачає наради, під час яких перевіряється рецензентами вихідний код продукту разом з його дизайном згідно задокументованих вимог, часто в присутності автора коду для відповіді на поставлені запитання.

Інспекції проводяться для розширення початкових стандартів чи перевірки наявності раніше вказаних помилок.

4. Забезпечення відповідного робочого середовища для команди.

Позитивне робоче середовище неодмінно гарно впливає на продуктивність співробітників і їх ставлення до роботи. Існують цілі інструкції щодо способів створення комфортних умов праці.

5. Впровадження приймального тестування користувача (UAT).

У розробці продукту є особи користувача для визначення ідеального клієнта або типового користувача даного продукту. Це вигадані особи, які мають моделі поведінки та мету цільової аудиторії представленого продукту. Команда контролю якості використовують їх для пошуку помилок у програмному забезпеченні. Проте суттєвим недоліком є те, що неможливо представити весь спектр моделей поведінки вигаданих осіб. Тестування кінцевим користувачем відбувається зазвичай на завершальних етапах розробки програмного забезпечення, що допомагає виявити помилки, які до цього не виявлялися. Цей вид тестування відповідно до Usersnap має 5 типів UAT.

6. Оптимізація використання автоматизованих тестів.

Автоматизоване тестування або використання інструментів автоматизації для запуску тестів вартують уваги, оскільки цей метод дозволяє економити час, зменшувати кількість людських помилок, покращувати можливості тестування, проводити пакетне тестування і паралельне виконання. Існує велика кількість інструментів для автоматизації тестування, які можуть мати відкритий вихідний код і комерційний. Selenium, Katalon Studio, Unified Functional Testing, Test Complete, Watir - найпопулярніші з них, які варто перевірити в першу чергу. Автоматизоване тестування діє і в рамках традиційних робочих процесів Agile, і є частиною методології DevOps, і практики безперервної інтеграції. Безперервна інтеграція (CI) є практикою розробки, що інтегрує неодноразові зміни в продукт кілька разів на день. Кожен фрагмент коду проходить «інтеграційні тести» для швидкого виявлення помилок та баги та легшої їх локалізації. Виправданим для надійності коду є поєднання CI з автоматизованим тестуванням. Інструменти з відкритим вихідним кодом Bamboo, Hudson та Cruise Control уможливлюють впровадження безперервної інтеграції у необхідному середовищі.

Безперервна доставка (CD) вважається еволюційним методом, який дозволяє швидко випускати зміни для своїх клієнтів у сталий спосіб. CD дозволяє випускати нові частини коду за їх готовності. Зазвичай автоматично розгортається кожна зміна після свого тестування. Це є досягненням завдяки високому рівню автоматизації тестування та розгортання. Оскільки практики CI та CD вимагають безперервного тестування, це виводить автоматизацію тестування на новий рівень.

7. Впровадження дослідницького та спеціального тестування.

Незважаючи на явні переваги автоматизації тестування, існують певні її обмеження в разі розгляду продукту з точки зору користувача. В цьому випадку використовуються інші методи тестування. Основою дослідницького та спеціального тестування є людська творчість, майже без

наявності документації, характерною є обмеженість або взагалі відсутність планування, обидва дещо випадкові, працюють для виявлення незвичайних дефектів, які не ідентифікуються іншими, структурованими, тестами. Дослідницьке тестування займається дослідженням продукту без заздалегідь визначених тестових кейсів для того щоб виявити, як насправді цей продукт працює. Таке виявлення помилок вимагає від тестувальників відповідного досвіду, професійної інтуїції та уяви, проводиться «на льоту» при негайній розробці і майже одночасному виконанні тесту. Цей спосіб тестування передбачає випробування різних реальних сценаріїв і поведінки користувачів, що допомагає швидко оцінити систему з отриманням негайного зворотнього зв'язку та відслідкувати області для подальшої діагностики. Проблемою дослідницького тестування є його документування з відтворенням збоїв та повідомленням про дефекти через відсутність запланованих сценаріїв і структурованості.

Спеціальне тестування є більш спонтанним та менш формальним методом, що грунтується на техніці вгадування помилок. Таке тестування проводиться після всіх інших тестів, його основною перевагою є швидке впровадження без ніякої підготовки і структурованої процедури. Такого роду хаотична перевірка може виявити дефекти, які не виявляються під час формальних тестів, а її результати непередбачувані і випадкові.

Попри те, що ці два методи мають багато спільного, є й деякі відмінності, адже спеціальне тестування передбачає ознайомлення з представленим програмним забезпеченням заздалегідь; а при дослідницькому - тестувальник дізнається про продукт лише під час його тестування. Спеціальне тестування виконується зазвичай вже на кінцевому етапі розробки після формального тестування, а дослідницьке тестування - будь-коли під час спринтів. Найкращим є доповнення автоматизованого тестування дослідницьким та спеціальним тестуванням для збільшення покриття тестування та покращення користувацького досвіду.

8. Використання вимірювання якості коду.

Виділяють такі важливі аспекти якості програмного забезпечення: надійність, ефективність роботи, безпека, зручність супроводу та швидкість доставки. Надійність визначає тривалість роботи системи без збоїв. Метою перевірки надійності є зменшення часу простою програми. Ефективність роботи визначає здатність системи виконувати будь-яку дію протягом заданого інтервалу часу. Ефективність можна виміряти за допомогою таких метрик, як навантажувальне тестування, що дає розуміння верхньої межі продуктивності системи та занурювальне тестування, яке перевіряє тривалість певного навантаження і визначає, коли продуктивність почне погіршуватися.

Безпека дбає про здатність системи захищати інформацію від ризику програмних порушень та запобігання втрати інформації.

Супроводжуваність передбачає здатність системи до модифікації програмного забезпечення, його адаптації для інших цілей, передачі його іншій команді розробників, супровід відповідності новим бізнес-вимогам.

Швидкість доставки програмного забезпечення є важливим аспектом, оскільки кількість випусків програмного забезпечення є основним показником того, як часто нове програмне забезпечення надається користувачам.

9. Ефективні звіти про помилки.

Професійно написаний баг-звіт з чітко визначеною проблематикою робить тестування програмного забезпечення більш ефективним, одразу спрямовуючи інженерів на усунення негараздів. Це свого роду наріжний камінь комунікації між QA спеціалістом та розробником, адже неякісний звіт може породити серйозні непорозуміння. Основними рекомендаціями щодо написання баг-звіту є надавання рішень якщо вони можливі, відтворення помилки перед повідомленням про неї, повідомлення про баг, який можна відтворити з подачею чіткої покрокової інструкції для його відтворення. Обов'язковим є вказування контексту і уникнення інформації, що може мати іншу інтерпретацію. Якщо відбувається періодичне відтворення багу, то про це потрібно повідомляти. До того ж звіт про помилку має бути чітким для того, щоб розробники могли зрозуміти збій, разом з інформацією про те, що бачить QA і що очікують побачити. Має бути детально описано, що пішло не так, і вирішення лише однієї проблеми в одному завданні. Спрощує роботу інженерів додача скріншотів прикладів збоїв, які висвітлюють дефект.

10. Отримання максимальної віддачі від інструментів управління тестуванням.

Інструментами управління тестуванням є програмні продукти за допомогою яких команди QA структурують та керують процесом тестування. Такі платформи можуть інтегруватися з системами автоматизації тестування, інструментами CI/CD, інструментами відстеження помилок та іншими програмними рішеннями компанії. Такі програмні продукти можуть планувати діяльність з тестування, містити фіксування вимог та інформацію про результати тестування, створювати тестові кейси та керувати їх середовищами, створювати звіти про виконання тестів, забезпечувати наочність, здійснювати обмін даними між членами команди. На ринку існує широкий вибір інструментів управління тестуванням для різних потреб і бюджетів.

Висновки

У статті представлено методи управління якістю програмного забезпечення. Серед методів вдосконалення процесу тестування програмного забезпечення та підвищення якості програмних продуктів можна виділити: планування процесу тестування та контролю якості, використання орієнтованого на тестування управління розробкою програмного забезпечення, проведення офіційних технічних оглядів, забезпечення відповідного робочого середовища для команди, впровадження приймального тестування користувача (UAT), оптимізація використання автоматизованих тестів, впровадження дослідницького та спеціального тестування, використання вимірювання якості коду, ефективні звіти про помилки, отримання максимальної віддачі від інструментів управління тестуванням. Предсталена методика дозволяє стверджувати, що для ефективного управління якістю програмного забезпечення необхідно охоплювати всі його ключові аспекти, як то ефективне планування, орієнтований на тестування підхід до управління якістю та спеціально сформовану професійну команду.

Література

1. Павленко, М. А., Осієвський, С. В., & Данюк, Ю. В. (2021). Методологічні основи підвищення якості програмного забезпечення інтелектуальних систем прийняття рішення. *Системи обробки інформації*, (1(164)), 55–64. https://doi.org/10.30748/soi.2021.164.06

2. Катаєва, Є., Одокієнко, С., Люта, М., & Савченко, Я. (2020). Практичний аналіз якості програмного забезпечення з відкритим кодом. *Управління розвитком складних систем*, (44), 49–55. https://doi.org/10.32347/2412-9933.2020.44.49-55

3. Грицюк, Ю. І. (2022). Система управління якістю програмного забезпечення. *Український журнал інформаційних технологій*, 4(1), 1–20. https://doi.org/10.23939/ujit2022.01.001

4. Yakovyna, V. S., & Symets, I. I. (2021). Towards a software defect proneness model: Feature selection. *Applied Aspects of Information Technology*, 4(4), 354–365. https://doi.org/10.15276/aait.04.2021.5

5. Imran, M., Cortellessa, V., Di Ruscio, D., Rubei, R., & Traini, L. (2024, June). An empirical study on code coverage of performance testing. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1–10). https://doi.org/10.1145/3661167.3661196

6. International Organization for Standardization. (1986). *ISO 8402:1986 Quality — Vocabulary*. https://www.iso.org/standard/15570.html

7. International Organization for Standardization. (2011). *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. https://www.iso.org/standard/35733.html

References

1. Pavlenko, M. A., Osiievskyi, S. V., & Daniuk, Yu. V. (2021). Metodolohichni osnovy pidvyshchennia yakosti prohramnoho zabezpechennia intelektualnykh system pryiniattia rishennia. Systemy obrobky informatsii, (1(164)), 55–64. https://doi.org/10.30748/soi.2021.164.06

2. Kataieva, Ye., Odokiienko, S., Liuta, M., & Savchenko, Ya. (2020). Praktychnyi analiz yakosti prohramnoho zabezpechennia z vidkrytym kodom. Upravlinnia rozvytkom skladnykh system, (44), 49–55. https://doi.org/10.32347/2412-9933.2020.44.49-55

3. Hrytsiuk, Yu. I. (2022). Systema upravlinnia yakistiu prohramnoho zabezpechennia. Ukrainskyi zhurnal informatsiinykh tekhnolohii, 4(1), 1–20. https://doi.org/10.23939/ujit2022.01.001

4. Yakovyna, V. S., & Symets, I. I. (2021). Towards a software defect proneness model: Feature selection. Applied Aspects of Information Technology, 4(4), 354–365. https://doi.org/10.15276/aait.04.2021.5

5. Imran, M., Cortellessa, V., Di Ruscio, D., Rubei, R., & Traini, L. (2024, June). An empirical study on code coverage of performance testing. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 1–10). https://doi.org/10.1145/3661167.3661196

6. International Organization for Standardization. (1986). ISO 8402:1986 Quality — Vocabulary. https://www.iso.org/standard/15570.html

7. International Organization for Standardization. (2011). ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/standard/35733.html