УДК 004.415.3

ІНТЕГРАЦІЯ ПРАКТИК ЗАБЕЗПЕЧЕННЯ ЯКОСТІ У ЖИТТЄВИЙ ЦИКЛ ВЕБ-ЗАСТОСУНКУ ЯК СКЛАДОВА СТРАТЕГІЇ НАДІЙНОСТІ ТА СТІЙКОГО РОЗВИТКУ ПРОДУКТУ

Браташ С.П.
Національний університет «Львівська політехніка»
Україна, Львів
solomiia.p.bratash@lpnu.ua

Анотація. *У роботі розглянуто підхід до інтеграції забезпечення якості веб-застосунків на всіх етапах життєвого циклу розробки. Встановлено, що якість має розглядатися як спільна відповідальність команди та невід'ємна частина процесу планування, розробки, деплойменту й подальшого моніторингу. Описано вплив безперервного тестування, автоматизованих перевірок коду, CI/CD і збору метрик на стабільність релізів, зменшення технічного боргу та скорочення вартості усунення дефектів. Наведено аргументацію, яким чином така модель формує культуру якості та створює умови для прогнозування ризиків.*
Ключові слова: *забезпечення якості; веб-застосунок; CI/CD; автоматизоване тестування; метрики якості; технічний борг; стабільність релізу.*

Вступна частина. Забезпечення якості веб-застосунків в умовах змін вимог користувачів і ринку вимагає переходу від підходу, де тестування розглядається як фінальний етап перед релізом, до підходу, у якому якість є наскрізною характеристикою всього життєвого циклу продукту. Сучасні організації працюють у режимі постійної доставки функціональності, де швидкість випуску нових версій безпосередньо впливає на конкурентоспроможність. У такому середовищі помилка, виявлена вже після розгортання програмного забезпечення, коштує значно дорожче і несе операційні ризики. Тому критично важливо інтегрувати перевірку якості у кожен етап процесу розробки: від планування та аналізу вимог до підтримки вже розгорнутого рішення.

Мета дослідження. Метою дослідження є обґрунтування моделі, у якій забезпечення якості веб-застосунку інтегрується у спринт як безперервний процес, а також демонстрація ключових переваг цього підходу для бізнесу, команди розробки та довгострокової стабільності продукту. Додатковою метою є уточнення ролі командних практик — таких як Code Review, Static Code Analysis, CI/CD, автоматизоване тестування та моніторинг продуктивного середовища — у зменшенні технічного боргу, підвищенні надійності релізів і формуванні культури відповідальності за якість.

Основна частина. Інтегрований підхід до забезпечення якості веб-застосунків ґрунтується на принципі, за яким якість програмного продукту розглядається не як завершальний етап життєвого циклу розробки, а як наскрізна характеристика всіх його фаз, починаючи з формулювання вимог і завершуючи експлуатацією програмного забезпечення у продуктивному середовищі. Традиційна каскадна модель, у рамках якої тестування виконується після основних робіт з імплементації, призводить до того, що дефекти виявляються на пізніх стадіях, коли їхнє виправлення є витратним, трудомістким і технічно ризиковим, оскільки будь-яка корекція коду на фінальному етапі може спричинити непередбачувані зміни в інших компонентах системи. На противагу цьому, у сучасних підходах забезпечення якості перевірка та контроль якості реалізуються як безперервний

процес. Кожна зміна коду супроводжується автоматизованими перевірками, статичним аналізом, лінтингом, автоматизованим прогоном тестів, а після розгортання — спостереженням за поведінкою системи в реальному середовищі експлуатації. Зокрема, автоматизоване тестування дає змогу уникнути надмірної залежності від повторюваних ручних сценаріїв, оскільки однотипні дії не потребують багаторазового відтворення вручну, а тестові випадки виконуються на відокремленому, відтворюваному середовищі із контрольованим станом даних, що забезпечує стабільність і повторюваність результатів перевірки [1]. Така організація процесу уможливлює не лише виявлення дефектів на ранніх стадіях, але й формування прогнозованої поведінки системи під час подальших ітерацій розробки.

Вагомою перевагою інтегрованого підходу до якості є можливість підвищення частоти випуску стабільних релізів без деградації експлуатаційних характеристик. У сучасних конвеєрах безперервної інтеграції та постачання (CI/CD) кожне оновлення коду у репозиторії автоматично ініціює виконання статичного аналізу, лінтингу й модульних тестів для раннього виявлення синтаксичних і логічних помилок. Далі відбувається складання застосунку з перевіркою залежностей, інтеграційне тестування взаємодії між модулями та програмними інтерфейсами, а перед розгортанням у продуктивне середовище здійснюються додаткові перевірки на предмет продуктивності, безпеки та відповідності політикам якості. Після розгортання здійснюється безперервний моніторинг поведінки системи в режимі реального часу з можливістю ініціювання контрольованого відкату у разі виявлення відхилень від очікуваних характеристик продуктивності або стабільності [2]. Таким чином відбувається перехід від епізодичних, великомасштабних, високоризикових випусків до коротких, контрольованих ітерацій постачання, що знижує як технічні, так і операційні ризики. Інтеграція інструментів керованої автоматизації збірки, тестування та розгортання у межах уніфікованого середовища, зокрема на базі хмарних сервісів, таких як Azure Pipelines, додатково сприяє подоланню організаційних і технологічних обмежень, пов’язаних із масштабуванням, залежностями та узгодженням інструментів. Вказані сервіси пропонують цілісну підтримку життєвого циклу постачання, включно з автоматизованою збіркою, перевіркою якості, розгортанням, контролем відповідності нормативним вимогам і політикам безпеки, а також мінімізацією кількості ручних дій, що знижує кількість людських помилок і підвищує відтворюваність процесу розгортання [3]. Таким чином забезпечується не лише прискорене постачання функціональності, але й сталість процедур контролю якості на кожній стадії.

Важливим наслідком описаної моделі є зниження вартості усунення дефектів. Виявлення помилок на ранніх стадіях розробки — на рівні окремого модуля або інтеграції між компонентами — дає змогу уникнути ситуації, коли дефект проникає у продуктивне середовище, впливає на реальні дані і вимагає термінового втручання команди підтримки. Така превентивність напряму впливає на зменшення технічного боргу, оскільки корекція здійснюється не у формі виправлення вже розгорнутих версій, а у формі контрольованих змін на рівні вихідного коду до введення його в промислову експлуатацію. Водночас поєднання автоматизованого тестування та експлуатаційного моніторингу після розгортання дає змогу не лише виявляти відмови і деградацію швидкодії, але й швидко локалізувати причину таких відхилень і застосувати механізми оперативного відкату, що мінімізує тривалість впливу інциденту на кінцевого користувача [2]. Такий підхід підвищує стійкість сервісу і трансформує функцію підтримки з реактивної (усунення інцидентів постфактум) на проактивну (запобігання інцидентам через технологічні запобіжники якості).

Ще одним ключовим структурним елементом інтегрованого підходу є колективний характер відповідальності за якість. У традиційних моделях розробки тестування часто позиціонується як окрема діяльність, що належить до сфери компетенції вузькоспеціалізованого підрозділу, тоді як розробники формально завершують реалізацію функціональності без гарантій її стабільності у продуктивному середовищі. У рамках сучасної моделі якість розглядається як спільна відповідальність усіх учасників процесу: розробники забезпечують підтримуваність і коректність змін за допомогою код-рев'ю; аналітики формують чіткі, верифіковані критерії прийняття, що мінімізує двозначність очікуваної поведінки; інженери автоматизації проєктують тестову інфраструктуру таким чином, щоб її можна було безперервно виконувати у конвеєрі збирання та постачання; інженери з експлуатації та інфраструктури відповідають за передбачуваність розгортання та за спостережуваність після випуску. Такий підхід узгоджується з сучасним трактуванням поняття «якість програмного продукту» як багатовимірної властивості, що охоплює не лише внутрішні характеристики коду, а й зовнішню придатність до використання, відповідність вимогам, стабільність функціонування та очікування кінцевого користувача. Згідно з підходами до моделювання якості програмних продуктів, якість визначається сукупністю властивостей системи та її відповідністю вимогам і «придатністю до використання», при цьому критерії якості є контекстно залежними і не можуть бути зведені до єдиного показника [4]. Таким чином, якість не може бути делегована одному учаснику процесу, оскільки вона водночас є технічною, операційною та бізнесовою категорією.

Окрему роль у підтримці якості відіграє систематичне накопичення та інтерпретація кількісних метрик. Зрілий процес розробки передбачає відмову від інтуїтивного оцінювання готовності на кшталт «система виглядає стабільною» і перехід до керованої аналітики, що охоплює показники інтенсивності дефектів, середнього часу їх усунення, покриття критичних шляхів автоматизованими тестами, кількості інцидентів після релізу та стабільності сервісу після змін. Ці метрики виконують не лише описову, але й прогнозну функцію, оскільки дають змогу ідентифікувати модулі з підвищеним ризиком деградації, планувати цілеспрямовані рефакторинги та коригувати пріоритети у плануванні робіт з урахуванням ризику для стабільності системи, а не лише бізнес-вартості функціональності. Такий підхід узгоджується з концепцією так званих метрик процесу (in-process metrics), у межах яких якість оцінюється під час виконання проєкту, а не лише після випуску. У рамках моделі «зусилля/результат» (Effort/Outcome Model) пропонується розглядати не тільки результативні індикатори, такі як інтенсивність виявлення дефектів і частота відмов, але й індикатори зусилля, зокрема прогрес тестування, ефективність тестового покриття, навантаження на середовище тощо. Узгоджене трактування цих двох груп показників дає змогу коректно інтерпретувати фактичний стан якості продукту ще до моменту його постачання у продуктивне середовище [4]. Важливо, що у такому випадку метрики використовуються не лише для звітності, а як інструмент управління ризиками та довгострокового планування розвитку архітектури.

Висновки. Сукупність описаних практик формує замкнений цикл зворотного зв'язку, у межах якого планування, імплементація, автоматизована перевірка, розгортання, операційний моніторинг і подальша адаптація процесів розробки взаємопов'язані і взаємно підсилюють одне одного. Такий цикл дозволяє організовувати роботу таким чином, що забезпечення якості стає вбудованою складовою кожної задачі у спринті, а не окремим етапом наприкінці. У практичному вимірі це означає, що під час планування ітерації одночасно враховуються вимоги до функціональності, вимоги до тестування, уточнення критеріїв прийняття, оновлення сценаріїв автоматизованого тестування, підтримка тестових середовищ,

підвищення спостережуваності в продуктивному середовищі та аналіз зібраних експлуатаційних показників. Таким чином якість перестає бути виключно технічним аспектом і набуває статусу стратегічного активу, який зменшує вартість усунення дефектів, стримує зростання технічного боргу, підтримує високу частоту та передбачуваність випусків, а також забезпечує стабільність і керованість розвитку веб-застосунку протягом усього його життєвого циклу [3].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Bangare, S. L., Borse, S., Bangare, P. S., & Nandedkar, S. (2012). Automated API testing approach. Department of Information Technology, STES's Sinhgad Academy of Engineering, Pune, Maharashtra, India. ISSN: 0975-5462.
[2] Jain, V. (2023). Continuous testing in CI/CD pipelines. *International Journal of Innovative Research and Creative Technology*. ISSN: 2454-5988. https://doi.org/10.5281/ZENODO.14883221
[3] Soma, V. (2024). Enhancing CI/CD pipelines with Azure Pipelines. *Journal of Engineering and Applied Sciences Technology*. ISSN: 2634-8853.
[4] Lazić, L. (n.d.). Software quality & testing metrics. *Recent Advances in Information Science. Proceedings of the 7th European Computing Conference (ECC '13),* 415–423. State University of Novi Pazar, Serbia. ISBN: 978-960-474-304-9.

Bratash S.P.

Integration of quality assurance practices into the web application lifecycle as a component of product reliability and sustainable development strategy

Abstract. *The paper examines an integrated approach to software quality assurance in web applications across all stages of the development life cycle. It argues that quality should be treated as a shared responsibility of the entire team and built into planning, implementation, deployment, and production monitoring rather than deferred to final testing. The study describes how continuous testing, automated code checks, CI/CD practices, and systematic metrics collection improve release stability, reduce technical debt, and lower the cost of fixing defects. It is shown that this model not only increases delivery speed and reliability but also enables risk prediction and supports a long-term quality culture within the team.*

Keywords: *quality assurance; web application; CI/CD; automated testing; quality metrics; technical debt; release stability.*