DOI: https://doi.org/10.32782/2524-0072/2024-64-74

УДК 004.052

ОЦІНКА НЕОБХІДНИХ РЕСУРСІВ І ЗУСИЛЬ ПРИ УПРАВЛІННІ ЯКІСТЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ПРИКЛАДІ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ У СКЛАДНИХ ТА СПЕЦИФІЧНИХ ПРОЕКТАХ

ESTIMATION OF REQUIRED RESOURCES AND EFFORTS IN SOFTWARE QUALITY MANAGEMENT ON THE SAMPLE OF PERFORMANCE TESTING FOR THE COMPLEX AND SPECIFIC PROJECTS

Ільницький Ігор Олегович
магістр з менеджменту організацій,
Івано-Франківський національний технічний університет нафти і газу
ORCID: https://orcid.org/0009-0000-0464-8275

Ilnytskyi Ihor
Ivano-Frankivsk National Technical University of Oil and Gas

Забезпечення якості є ключовою, невід'ємною та визначальною діяльністю у процесі розробки програмного забезпечення. Крім найкращого виконання усіх функціональних вимог до програмного продукту, важливим також є забезпечити його інші сторони, якими часто нехтується на початку життєвого циклу ПЗ: належний рівень безпеки та продуктивності, відповідний дизайн тощо. Іноді виконавець і клієнт нехтують виконанням перевірки якості продукту на відповідність нефункціональним вимогам або ще гірше не мають відповідних належних знань, а інформація, як правильно виконати замовлення, що відповідає критеріям, які не відносяться до функціоналу, знаходиться у індивідуальному користуванні компаній, які пройшли такий шлях на власній практиці спроб і помилок. В зв'язку з цим, під час користування програмами в реальному часі, можуть виникати несподівані помилки та виявлятись недоліки в роботі програм, на усунення яких не буде достатньо часу, що також призведе до використання додаткових грошових коштів. Крім того, в зв'язку з тим, що потрібен визначений час, це може призвести додатково до втрати клієнтів і, як наслідок, зменшення довіри до компаній та їх збитків. Виконавець замовлення має володіти належною експертизою в наведеній сфері, щоб оцінити необхідні ресурси та зусилля, знайти оптимальні рішення для забезпечення якості майбутнього продукту, а також має вміти обгрунтувати наведене клієнтові, щоб задовольнити кінцевого споживача і спільно досягти успіху. Якщо діяти на випередження, щоб своєчасно виконати завдання по забезпеченню нефункціональних вимог, можна значно зменшити витрати, зусилля і час необхідні для виконання цих завдань, адже процес є довготривалим, складним та дороговартісним. Для цього у дослідженні подається перелік з описом ключових аспектів забезпечення якості продукту, які необхідно проаналізувати і адаптувати (доповнити, змінити), щоб визначити та оцінити необхідні ресурси і зусилля на початкових стадіях SDLC, а також для прийняття відповідних управлінських рішень. Дана стаття базується на прикладі оцінки забезпечення належної продуктивності ПЗ у складних та специфічних проектах.

Ключові слова: менеджмент, експертиза, продуктивність, оцінка ресурсів та зусиль, забезпечення якості програмного забезпечення, управління якістю, нефункціональні вимоги, AQA, SDET, SDLC.

Quality assurance is the key, inalienable and determining activity in the process of the software development. Apart from the best implementation of the functional requirements of an application, it is also important to ensure other aspects: respective level of security and performance, concordant design. Sometimes the executor and client neglect to execute checks of the product quality for compliance with non-functional requirements or even worse do not have proper knowledge; furthermore, information about how to correctly implement an order, meeting criteria, that do not belong to the functionality, is in the individual use of companies that have gone through this path on their own practice of trial and error. In connection with the above mentioned, unexpected errors may occur and there may be detected imperfections in the working of programs to eliminate which there will be not enough time, which will also lead to the additional expenses. Moreover, because determined time is required, it may additionally lead to losing of clients and in result to reduction of trust in companies and their losses. The executor of the order must have

the appropriate expertise in the given field to estimate the necessary resources and efforts, find optimal solutions to ensure the quality of the future product, and must also be able to justify the given to the client in order to satisfy the end user and achieve success together. Acting ahead of time to complete non-functional requirements tasks in a timely manner can significantly reduce the costs, efforts, and time required to complete these tasks, as the process is lengthy, complex, and expensive. For this, the study provides a list and its detalization with the key aspects of product quality assurance, which must be analyzed and adapted (supplemented, changed) to identify and estimate the necessary resources and efforts at the initial stages of the SDLC as well as for making appropriate management decisions. This article is based on the example of ensuring the proper software performance level in complex and specific projects.

Keywords: management, expertise, performance, resource and effort estimation, software quality assurance, quality management, non-functional requirements, AQA, SDET, SDLC.

Постановка проблеми. У розробці програмного забезпечення, зокрема у складних, комплексних проектах, забезпечення високої якості ПЗ, залишається найважливішим викликом, як для виконавця так і замовника. Компанія-розробник програмного забезпечення має бути в силі належно і завчасно оцінити необхідні ресурси та зусилля для своєчасного забезпечення належної якості, запропонувати оптимальні рішення. Загальновідомо, що замовник в першу чергу зацікавлений отримати ту ж якість за менші кошти, але виконавець замовлення має вміти обґрунтувати та донести найвигідніші рішення для обох сторін з урахуванням якості кінцевого продукту, які на перший погляд можуть здаватися дорожчими для клієнта. Тому, виходячи з вищенаведеного, оцінка ресурсів та зусиль є першочерговою та найважливішою складовою, яка впливає на всі подальші процеси і успіх проекту.

Аналіз останніх досліджень і публікацій. З власного досвіду, комунікації з іншими спеціалістами сфери інформаційних технологій і на основі аналізу різних джерел, як правило, розробка і забезпечення якості програмного забезпечення зосереджуються першочергово на виконанні функціональних вимог наданих замовником. Часто активності по забезпеченню продуктивності, безпеки та належного дизайну ПЗ стають другорядними або взагалі не є в пріоритеті, а відповідні завдання можуть несподіваним чином з'явитись на пізніших етапах життєдіяльності продукту. Наведені активності є нетривіальними, вимагають належних експертних знань, є тривалими по часу та дороговартісними.

В дослідженнях та публікаціях, на форумах інтернет сайтів, у спеціалізованій літературі тощо є багато теоретичної та практичної інформації з розробки та забезпечення якості, і зокрема щодо продуктивності ПЗ.

Важливим є консолідувати та описати ключові аспекти та як провести оцінку необхідних ресурсів та зусиль при управлінні якістю програмного забезпечення на прикладі забезпечення його належної продуктивності у складних та специфічних проектах, таких які відповідають наступним критеріям:

1) складна архітектура, понад 100 мікросервісів;
2) багато учасників: команди розробників різних частин комплексного продукту та менеджмент знаходяться на стороні замовника, виконавця, третіх сторін, також є сторонні сервіси;
3) різнотипні інтеграції – file-, api-based;
4) різнодоступні інтеграції – повністю- та частково-доступні;
5) необхідність застосування кількох способів для навантаження системи – за допомогою тестового фреймворку (api-based інтеграції), скриптів для генерації файлів великого розміру та з великим об'ємом даних (file-based інтеграції) вручну або автоматизовано (залежно від правил security), при участі розробників за межами компанії-виконавця (частково доступні file-based інтеграції);
6) значний перелік джерел всередині і за межами компанії з прямою і опосередкованою інформацією різного формату для аналізу і консолідування результатів тестування продуктивності програм, на відміну, коли всі результати доступні в одному звіті (reporting);
7) складна комунікація: працівники з різних країн – України, Британії, США, Індії тощо (наприклад різні часові пояси і культура), компанії відрізняються по технологіях, графіках роботи, рівню експертизи, ієрархічному складу персоналу і його поділу, цілях – кожен працює над своєю індивідуальною частиною продукту тощо;
8) різний рівень або відсутність різнопланової експертизи по темі.

Виділення невирішених раніше частин загальної проблеми. Як згадано раніше, є чимало джерел інформації по питанню забезпечення якості програмного забезпечення

з точки зору його продуктивності. Наведені ресурси надають теоретичну і практичну інформацію з різними прикладами, висвітлюють з різних ракурсів окремі питання пов'язані з тестуванням продуктивності ПЗ. Важливим є дати змогу зацікавленим сторонам побачити загальну картину і висвітлити аспекти, на які слід звернути увагу, щоб було можливим провести комплексну оцінку необхідних ресурсів і зусиль для виконання високоякісного замовлення.

Мета статті. Дана робота зацікавить менеджерів різного рівня управлінського складу для прийняття необхідних управлінських рішень (бюджети, ресурси, зусилля, персонал, технології, інструменти, необхідність тощо), а також спеціалістів, задіяних у розробці і забезпеченні якості продукції. Основною метою даної роботи є надати приклад комплексної оцінки ресурсів і зусиль (estimate) для належного забезпечення якості програмного забезпечення на прикладі тестування його продуктивності, для складних і специфічних проектів. Дослідження є базою та може доповнюватись, оновлюватись і має бути адаптованим до індивідувальних проектів та методологій розробки. Може бути спільно використаним на стороні виконавця, замовника, третіх сторін, сторонніх сервісів.

Виклад основного матеріалу дослідження. Для виконання ефективного управління якістю програмного забезпечення, необхідно детально розглянути та оцінити наступні важливі ключові моменти (аспекти), для визначення необхідних ресурсів і зусиль по забезпеченню якості (продуктивності) програмного забезпечення і прийняття відповідних управлінських рішень.

Архітектура. Загальний опис проекту-прикладу: складний проект, 50 інтеграцій, 150 API-based та file-based мікросервісів, по 2-5 мікросервісів на інтеграцію. Розглядається на прикладі наступних технологій: microservices, Kubernetes, Java, Javascript, Kafka, PostgreSQL, Keycloak, AWS, AKHQ, Kibana, Grafana, Jira, Confluence, etc.

Приклад інтеграції наведено в наступній схемі (рис. 1).

Опис діаграми (схеми) виглядає наступним чином:

1) зовнішня програма або test framework (тестовий фреймворк) відправляє дані в kafka producer (продюсер);

2) producer обробляє та створює дані в kafka (сховище даних);

3) kafka consumer (консюмер) споживає, обробляє, зберігає (або не зберігає) оброблені дані з kafka та відправляє у зовнішню програму клієнта чи третіх сторін.

Також є інтеграції, що базуються на файлах (file-based). В такому випадку схема виглядає наступним чином:

1) producer отримує notification (нотифікацію) SQS Queue з AWS і зчитує новий файл з даними відповідно до нотифікації;

2) розділяє і перетворює дані отриманого файлу у окремі об'єкти, які зберігаються у kafka;

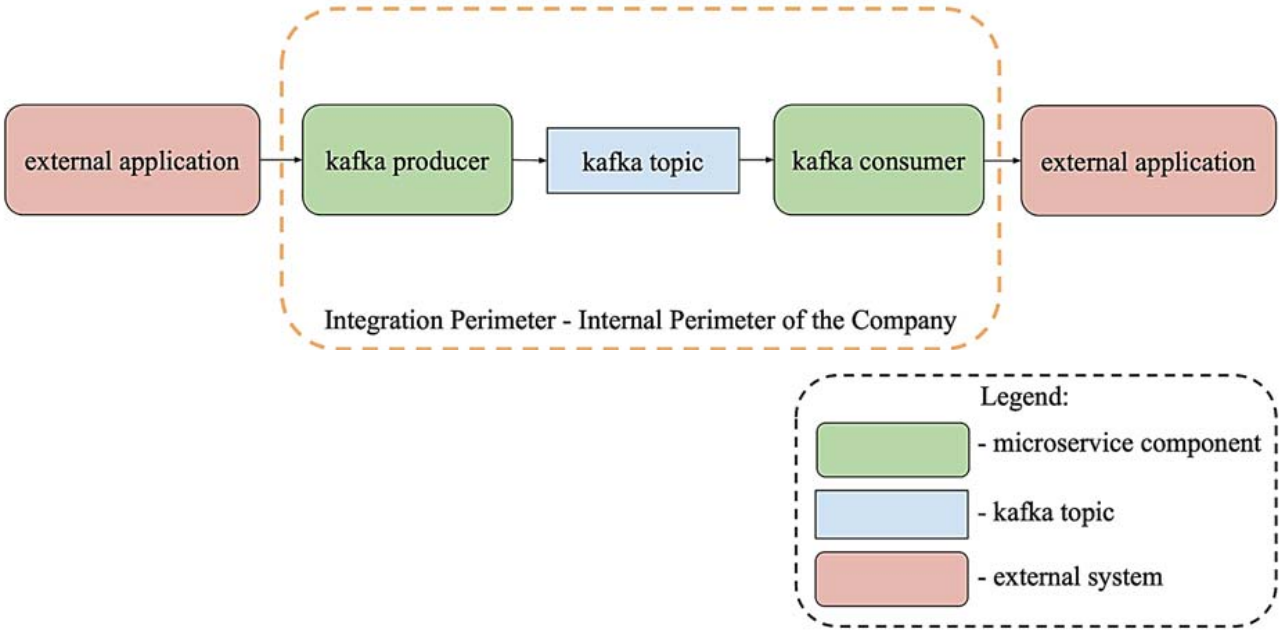

Рис. 1. Приклад дизайну однієї API-based інтеграції

3) kafka consumer (консюмер) споживає, обробляє, зберігає (або не зберігає) оброблені дані з kafka та відправляє у зовнішню програму клієнта чи третіх сторін.

File-based інтеграції додатково мають іншу наступну реалізацію:

1) мікросервіс отримує notification (нотифікацію) SQS Queue з AWS щодо поступлення набору нових файлів з даними;

2) далі наведений мікросервіс проводить валідацію файлів: ім'я, розмір формат тощо;

3) копіює файли у інші відповідні папки, змінивши за потреби відповідно до правил імена, і вносить інформацію у необхідну таблицю бази даних (PostgreSQL).

Поділ інтеграцій по типу і доступності всередині компанії:

1) повністю-доступні API-based інтеграції, коли дані відправляються автоматично за допомогою тестовго фреймворку;

2) повністю-доступні file-based інтеграції, коли дані надсилаються у вигляді файлів (csv, txt, xls, xml, pdf або іншого формату), створених за допомогою скрипта (наприклад на Javascript), автоматично чи вручну (залежно від вимог security і доступів та інших причин) на сервер (AWS) і далі зчитуються мікросервісом (продюсером);

3) частково-доступні з певних причин file-based інтеграції, коли дані зчитуються зі стороннього сховища; в такому разі тестування відбувається з залученням спеціалістів клієнта чи третіх сторін.

Доступність мікросервісів також впливає на необхідні зусилля по забезпеченню якості програмного забезпечення.

Доступна команді технічна документація (дизайн кожної інтеграції) оформляється розробниками на Confluence якнайшвидше, оскільки розблоковує виконання завдань по забезпеченню якості програмного забезпечення. Надаються доступи до необхідної документації клієнта.

Процес. Етапи розробки програмного забезпечення, включаючи забезпечення якості, на прикладі наведеного вище дизайну будуть виглядати наступним чином:

1) розробка мікросервісів (компонентів);

2) функціональне тестування мікросервісу (може включати зовнішню комунікацію з клієнтом, третіми сторонами);

3) тестування продуктивності мікросервісу (може включати зовнішню комунікацію з клієнтом, третіми сторонами);

4) функціональне тестування інтеграції (включає зовнішню комунікацію з клієнтом, третіми сторонами);

5) тестування продуктивності інтеграції (включає зовнішню комунікацію з клієнтом, третіми сторонами);

6) одночасне тестування набору інтеграцій, обраних за певною логікою, наприклад спільний кінцевий endpoint;

7) системне тестування інтелектуального продукту компанії;

8) спільне тестування рішення з усіма сторонами;

9) прийняття рішення про належну якість (спільного) продукту.

Спрощену схему розробки програмного забезпечення зображено на наступному рисунку (рис. 2).

Відповідні активності вищенаведених етапів розробки програмного забезпечення – розробка, функціональне тестування (мікросервісу, інтеграції), тестування продуктивності (мікросервісу, інтеграції, набору інтеграцій, системне, спільне), а також повторне тестування (retesting), регресійне тестування (regression testing) тощо – будуть виконуватись як послідовно, так і, по мірі зрілості продукту, все більше і більше паралельно. Коли тестується ціла інтеграція, kafka producer покривається тестами повторно, але такий підхід дає змогу швидко отримати результати – вже після першої перевірки продюсера. Кожен мікросервіс, інтеграція можуть і будуть перевірятись багаторазово (5-10 разів) в зв'язку з виправленням помилок і/або через зміни, та при комплексному і спільному тестуванні. Слід розуміти, що клієнт і треті сторони також потребуватимуть допомоги у перевірці своєї частини продукту по мірі прогресу, що вимагатиме додаткових комунікацій і активностей зі сторони компанії, яка проводить оцінку (estimation). Залучення зовнішніх сторін необхідне найперше для відправки даних та для проведення аналізу і надання інформації щодо отриманих ними даних (reconciliation), а також, по-друге, для тестування їхньої частини продукту. Тут навмисне зазначається "по-друге", тому що тестування своїх продуктів різними сторонами, по мірі готовності, проводиться, в першу чергу, індивідуально, незалежно, оскільки темпи і графіки (етапи) виконання, технології, дизайни, підходи, цілі та інше відрізняються, що ускладнює синхронізацію дій та робить її радше недоречною, так як скоріше буде заважати та обмежувати

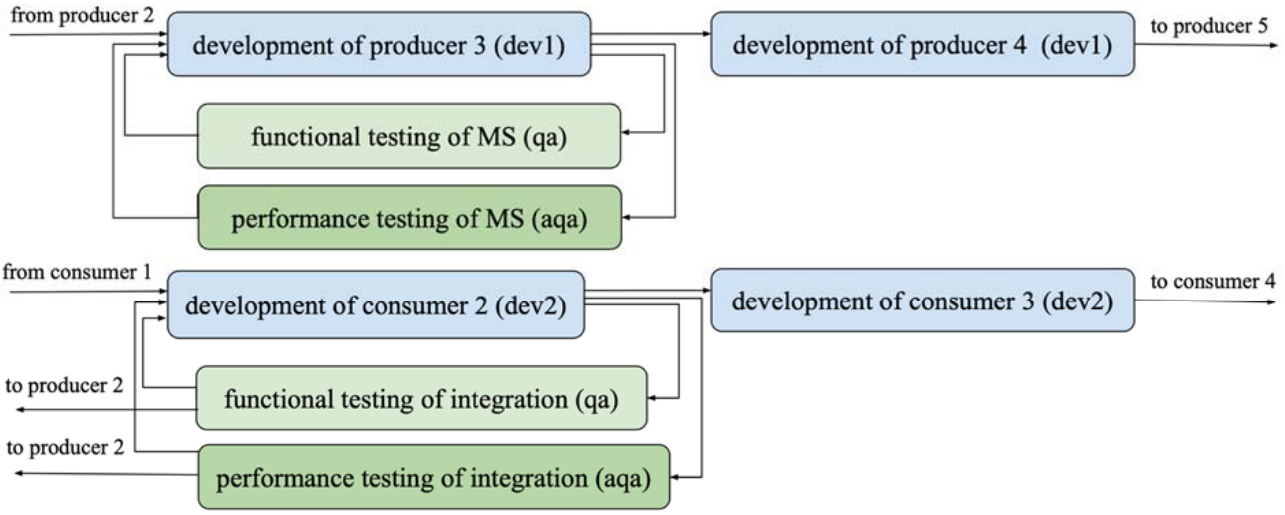

Рис. 2. Схематичне зображення етапів розробки програмного забезпечення

спеціалістів у виконанні їхніх завдань (пробне тестування, безпосереднє тестування, повторне тестування, регресійне тестування тощо). На прикінцевих етапах проводиться спільне тестування цілого рішення разом з усіма сторонами. Кожен раз завчасно інформується зацікавлені та відповідальні сторони визначеними каналами комунікації, та конфігурується середовище – збільшуються об'єми пам'яті, відключаються мікросервіси чи інтеграції (за допомогою відповідних feature switch), які не мають приймати участь у навантаженні. Сторони інформуються знову після завершення тесту, зменшуються ресурси, видаляються дані (змінюється offset), включаються інтеграції.

Дані. Ще однією важливою складовою оцінки є отримання та аналіз статистичних даних по інтеграціях, які мають надаватися замовником у вигляді таблиць та відповідної документації, за рік (краще кілька) з деталями навантажень по кожній (онлайн обробка даних чи по графіку – раз на день, кілька разів на тиждень тощо) та годинах (або більш детальніше) для визначення пікових навантажень. Необхідно також врахувати тривалість робочого дня (коли саме користувачі відвідують сайт), години чи ресурси, якими можна нехтувати (не бажано) через незначні навантаження; не слід зменшувати тестові навантаження на об'єми знехтуваних годин чи ресурсів, а включити в розрахунок чи розподілити між іншими сценаріями. Важливо зазначити, що в кінцевому результаті весь функціонал сайту замовника, має бути покрито тестами, не тільки проект-приклад. Важливим також є знання про потенційне збільшення навантажень в майбутньому в зв'язку з популяризацією функціоналу зі стихійних чи спланованих причин (маркетинг). Підраховані навантаження, на основі оцінки наданих даних, у сценаріях тестування збільшуються традиційно на 30-50% чи навіть в два-три рази з цим коефіцієнтом, оскільки можливі крайні випадки, коли відбувається накопичення даних за 2 чи більше днів по причині специфіки даної ділянки продукту, технічних робіт, неполадок на різних сторонах тощо. Потрібно, щоб спеціалісти AQA/SDET мали знання зі статистики, для належного аналізу даних і визначення коректних навантажень по тестах, виходячи з максимальных нагрузок. Популярною помилкою є підрахунок усереднених навантажень виходячи з даних за день, місяць, рік, що діляться на відповідну кількість годин, бо таким чином можна не врахувати дійсні, а також потенційні пікові значення, як наслідок реальна різниця може бути вагомою.

Вимоги. Вимоги до якості мають бути чіткими та надані клієнтом, а у разі неточностей чи неясностей, уточнені в замовника. Вимоги можуть бути надані і/або запропоновані, та погоджені з замовником, і мають оформлятися у вигляді критеріїв прийомки програмного забезпечення. Чітке бачення вимог до спільного продукту на підготовчому етапі, зменшує кількість додаткових погоджень, дає ясні цілі і, як наслідок, береже час та кошти сторін.

Тестові сценарії. Для виконання перевірок на продуктивність важливо розуміти і виконати дизайн тестових сценаріїв. Сценарії перевірки продуктивності програм мають передбачати об'єми (кількість запитів на

секунду) та тривалість відповідних навантажень (типово 30-60 хв). Обов'язковими кроками є наступні перевірки:

Response time	час відповіді на запит	менше 1000 мс, інше
Successful requests	кількість успішних запитів	100 % або інше
Response body	актуальна відповідь	відповідає очікуваній

Будь-який сценарій також має дати відповіді на наведені далі запитання:

1. Чи достатньо виділено ресурсів?
2. Чи використовуються виділені ресурси ефективно?
3. Які помилки потрібно виправити?

Також проводиться звірка (reconciliation report) – кількість вхідних даних має відповідати кількості даних на виході:

Кількість позитивних сценаріїв / Quantity of happy pass scenarios	h	Кількість створених, прочитаних, оновлених і видалених даних / Quantity of created, read, updated and deleted data	crud
Кількість негативних сценаріїв / Quantity of negative scenarios	n	Кількість помилок і виключень / Quantity of errors and exceptions	ee
Загалом / Total	t	Загалом / Total	t

Інструменти. Вибір інструмента для тестування продуктивності програмного забезпечення здійснюється з існуючого або ж за допомогою так званого підходу "Proof of Concept" (PoC). Proof of Concept включає набір необхідних критеріїв, яким має відповідати інструмент для тестування продуктивності програмного забезпечення (обирається досвідченим AQA/SDET самостійно) і порівняння на основі цього 2-3 інструментів, для прийняття рішення щодо вибору одного з них. Вибір може бути зроблено наступним чином: обираються 3 засоби тестування, 1 може бути відсіяно на початковому етапі через технології на проекті чи інші причини, далі подається таблиця, де зазначається перелік критеріїв з описом чи і на скільки забезпечує кожен інструмент кожен критерій, а також чи потрібні постійні додаткові зусилля для підтримки засобу (наприклад у випадку відсутності звітності out of box). Додаткові зусилля приймаються як негативна сторона інструменту, а наприклад наявність звітності out of box – як позитивна. Обраний тестовий фреймворк використовується для API-based інтеграцій. У випадку file-based інтеграцій пишуться скрипти, наприклад на Java чи Javascript, для створення необхідної кількості файлів (csv, txt, xls, xml, pdf тощо) відповідного розміру і/або об'єму даних. Створені файли відправляються автоматично чи вручну, залежно від доступів, правил security тощо, що також враховується при оцінці. Також файли можуть відправлятись спеціалістами клієнта чи інших зовнішніх учасників, при відсутності доступних способів зробити це самостійно компанією-розробником.

Результати. Специфіка архітектури (приклади інтеграцій наведено) робить необхідним отримання даних з різних джерел, для аналізу і написання результатів тестування та прийняття управлінських рішень:

1. Звіт тестового фреймворку (test framework report).
2. Kafka (kafka-client, AKHQ).
3. Kibana.
4. Grafana.
5. Kubernetes.
6. Файли AWS, що генеруються по графіку (scheduling) спеціальними програмами-процесорами, які є складовими частинами деяких інтеграцій.
7. PostgreSQL.
8. Інформація, що надається клієнтом та третіми сторонами (дані, файли, сценарії тощо).
9. Інша інформація.

Тобто аналіз даних вимагає значних додаткових зусиль спеіцалістів, виходячи зі специфіки проекту, на відміну від простіших та зручніших випадків, коли всі результати тестування генерується інструментом автоматизації в одному звіті. Після аналізу отриманих даних, результати записуються у консолідований звіт із зазначенням відповідних статусів тестування (passed, failed, passed with comment). Результати мають додаватись та моніторитись постійно учасниками процесу.

Ліцензії. Важливо, щоб у AQA/SDET розробників була можливість виконувати тестові сценарії в необмеженій або в достатній зі значним запасом кількості. Якщо не враховувати

вищенаведений момент, у сторін виникає необхідність додаткових погоджень бюджетів з замовником чи третіми сторонами, що може призвести до вагомого збільшення часу виконання та доставки замовлення належної якості, на відміну від завчасного урахування необхідних ресурсів. Середовища розробки і тестування мають бути доступні постійно протягом запланованого часу.

Кадри. Важливими також є кількісний склад спеціалістів, їх компетенція, досвід, та необхідні знання для виконання поставлених завдань. Знання менеджера, розробника повинні включати не тільки зі сфери інформаційних технологій, а також обов'язково маркетингу, статистики, математики і сфери ведення бізнесу. Відсутність наведених додаткових знань призводить до необхідності постійно навчати виконавців та пояснювати свої дії іншим зацікавленим сторонам (без належних досвіду, кваліфікації, підготовки та знань), що є особливо критичним при складних проектах, коли участь в розробці продукту приймає кілька компаній (3 і більше), що також впливає на своєчасне забезпечення якості програмного забезпечення і його вартість. Безпосереднє виконання поставлених завдань (в тому числі оцінку) по забезпеченню якості програмного забезпечення мають виконувати AQA (automation quality assurance) engineers, SDET (software development engineers in tests). Вирішальне слово має бути за спеціалістами AQA/SDET при прийнятті рішень, що стосуються забезпечення якості продукту. Всі ці рішення мають привести в кінцевому результаті до оптимальних зусиль, термінів виконання продукту, його якості і вартості, мінімізують необхідність постійної відправки продукту на доопрацювання.

Комунікація. Команда розробників, менеджмент будуть комунікувати зі спеціалістами та менеджерами всередині компанії, і відповідними представниками інших компаній (клієнт, треті сторони, зовнішні сервіси) за допомогою наперед визначених внутрішніх та зовнішніх каналів комунікації, про що має бути домовлено завчасно. Всі важливі моменти повинні вестися і/або резюмуватися у письмовій формі. Комунікація має бути своєчасною, навіть на випередження.

Поділ праці, співпраця. Потрібно врахувати також, що окрім менеджерів і спеіцалістів на проекті, в компаніях також є інші команди (наприклад DevOps, команди, що теж використовують інфраструктуру для розробки, security тощо) і менеджмент різного рівня, з якими необхідно комунікувати та консультуватись, для успішного виконання завдань і досягнення поставлених цілей. Допоміжні команди виконуватимуть супутні завдання для виконання проекту, тому обговорення, прийняття рішень і постановка задач мають бути своєчасними.

Уточнення, запитання, відповіді. При виникненні запитань, необхідно без зволікань направити відповідний запит відповідальним спеціалістам, менеджменту. Всі запитання заносяться в таблицю із зазначенням теми листа, відповідальної персони (адресата), датою, деталями, статусом тощо.

Конфігурації. Продукт необхідно перевірити на різних конфігураціях середовища для вибору оптимальних варіантів, щоб міг працювати належним чином під довготривалим навантаженням, а також щоб уникнути перевитрат надлишково виділених ресурсів.

Підтримка. Сторони мають завчасно передбачити супровід програмного забезпечення, після виходу в production, оскільки це також частина процесу; клієнт може звернутись із запитом про доопрацювання частини замовлення, програми постійно змінюються та оновлюються, навантаження та параметри можуть змінитися теж.

Тестова стратегія, тестовий план, сценарії навантажень. По мірі повної оцінки вищенаведеного, тестова стратегія оновлюється відповідним розділом, пишеться тестовий план та сценарії навантажень. Документація може доповнюватися також рисунками для наочного зображення дизайну тестів: програми, шляхи проходження даних між ними, а також яким чином відбувається авторизація. Набір дизайн-схем має узагальнювати всі випадки інтеграцій, що підлягають тестуванню.

Бюджет, резервні кошти та ресурси. При початкових погодженнях і в подальшому, бюджет має включати резервні кошти, а також в компанії мають бути додаткові забезпечуючі ресурси (спеціалісти, інфраструктура тощо), щоб уникнути додаткових погоджень, оскільки це час і втрата темпу.

Висновки. Забезпечення якості програмного забезпечення з точки зору його продуктивності довготривалий, складний і дороговартісний процес, що потребує завчасних дій щодо оцінки необхідних зусиль і ресурсів для прийняття необхідних управлінських рішень. Є багато аспектів наведеної діяльності, описаних у статті, які необхідно дослідити та адаптувати всередині компанії, що виконує замовлення, та обґрунтувати клієнтові. Важливо

додатково зазначити, що існують проекти, для яких продуктивність програм є першочерговою. Непідготовлена компанія ризикує понести більші грошові та часові витрати, втратити імідж і клієнта. Джерела інформації по темі описують теоретичні та практичні аспекти з прикладами щодо тестування продуктивності з різних ракурсів, проте необхідною є консолідація всього набору інформації для проведення належної комплексної оцінки всього процесу забезпечення якості. Існують компанії, що мають належний рівень експертизи, досягнувши його методом проб і помилок, пройшовши величезний шлях по забезпеченню нефункціональних вимог програмного забезпечення, і в свою чергу самостійно користуються власно-розробленою методологією. Тематика складна, підлягає подальшому вивченню, вдосконаленню, розширенню, деталізації та адаптації до різних компаній і методологій розробки програмного забезпечення. Наявність експертизи впливає на формування хорошого психологічного клімату в команді і компанії загалом, сприяє своєчасному та якісному виконанню замовлення, є визначальною для іміджу та рентабельності підприємства, впевненості та задоволеності клієнтів і кінцевих споживачів, допомагає отримати нові замовлення, є основою для прийняття оптимальних управлінських рішень. Управлінцям рекомендую завчасно вивчити та впровадити в організації процес забезпечення продуктивності програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Luca Traini. 2022. Exploring Performance Assurance Practices and Challenges in Agile Software Development: An Ethnographic Study. *Empirical Software Engineering* 27, 3 (2022), 74. DOI: https://doi.org/10.1007/s10664-021-10069-3

2. Muhammad Imran, Vittorio Cortellessa, Davide Di Ruscio, Riccardo Rubei, Luca Traini. An Empirical Study on Code Coverage of Performance Testing. June 2024. Conference: EASE 2024: 28th International Conference on Evaluation and Assessment in Software Engineering. DOI:10.1145/3661167.3661196. URL: https://www.researchgate.net/publication/381513292_An_Empirical_Study_on_Code_Coverage_of_Performance_Testing

3. Shravan Pargaonkar (University of Texas at Arlington). A Comprehensive Review of Performance Testing Methodologies and Best Practices: Software Quality Engineering. November 2023. International Journal of Science and Research (IJSR) 12(8):2008-2014. DOI:10.21275/SR23822111402. URL: https://www.researchgate.net/publication/375450774_A_Comprehensive_Review_of_Performance_Testing_Methodologies_and_Best_Practices_Software_Quality_Engineering

4. Erik Whiting, Soma Datta. May 2021. Performance Testing and Agile Software Development: A Systematic Review. URL: https://www.researchgate.net/publication/351410867_Performance_Testing_and_Agile_Software_Development_A_Systematic_Review

5. Microsoft. J.D. Meier Carlos Farre Prashant Bansode Scott Barber Dennis Rea. 2007. Performance Testing Guidance for Web Applications (patterns & practices). URL: http://download.51testing.com/ddimg/uploadsoft/20101206/PerfTestGuide.pdf

6. Whittle, D. (2021). Continuous performance testing: Load testing in an Agile and DevOps world. Manning Publications.

7. Moore, S. (2023). Performance testing for developers: Increase your code quality with a practical approach to performance testing. Packt Publishing.

8. Zacharias, S. (2022). Performance testing in the cloud: Applications and tools. Apress.

9. Jain, A. (2021). Mastering performance testing: Unleash the power of JMeter, Gatling, and BlazeMeter. Packt Publishing.

10. Lelchuk, B. (2020). Practical performance testing: A guide for software product owners, developers, and QA engineers. Apress.