Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Покращення процесів розробки додатків баз даних за допомогою впровадження практик CI/CD

Виконав(ла): студент(ка) 6 курсу, групи САм-61
спеціальності 124 Системний аналіз
(шифр і назва спеціальності)

(підпис) Постолюк Т.М. (прізвище та ініціали)

Керівник (підпис) Липак Г.І. (прізвище та ініціали)

Нормоконтроль (підпис) Дуда О.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
"22" грудня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 124 Системний аналіз
(шифр і назва спеціальності)

студенту Постолюк Тарас Миколайович
(прізвище, ім'я, по батькові)

1. Тема роботи **Покращення процесів розробки додатків баз даних за допомогою впровадження практик CI/CD**

Керівник роботи **к.соц.-комунікац.н., доц. Липак Г.І.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від "*29*" *листопада* 2024 року № *4/7-1140*

2. Термін подання студентом завершеної роботи 19 грудня 2024 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП 1 ОГЛЯД ЛІТЕРАТУРИ ЗА ТЕМОЮ РОБОТИ 1.1 Опис методу визначення продуктивності конвеєра CI/CD 1.2 Тестування бази даних 2 ОСОБЛИВОСТІ CI/CD У ПРОГРАМАХ З БАЗАМИ ДАНИХ 2.1 Практики CI/CD у програмах баз даних 2.2 Проблеми впровадження CI/CD у програмах баз даних 2.3 Програмні засоби міграції та тестування баз даних 2.4 Технічні та організаційні передумови 3 ПОБУДОВА CI/CD-КОНВЕЄРУ З ДОДАТКАМИ БАЗ ДАНИХ 3.1 Проектування конвеєру 3.2 Обґрунтування архітектури конвеєру 3.3 Інфраструктура конвеєра CI/CD для програм баз даних 3.4 Випадки використання конвеєра CI/CD для програм баз даних 3.5 Варіант використання сховища даних (DWH) 3.6 Варіант використання внутрішньої бази даних (BE) 3.7 Логістика для роздрібної торгівлі на базі даних (RET) 3.8 Прийняття та оцінка конвеєра CI/CD 3.9 Кількісний аналіз від впровадження CI/CD 3.10 Якісний аналіз запровадження автоматизації 3.11 Вплив CI/CD 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 4.1 Методи та засоби психофізіологічного розвантаження як допоміжний процес в розробці ПЗ 4.2 Попередження аварій на виробництвах із застосуванням хлору ВИСНОВКИ ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С, к.т.н., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.11.24-15.11.24	*Виконано*
2.	Підбір наукових джерел по темі роботи	16.11.24-20.11.24	*Виконано*
3.	Переклад та опрацювання наукових джерел	20.11.24-23.11.24	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.11.24-10.12.24	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	11.12.24-12.10.24	*Виконано*
6.	Оформлення другого розділу	12.12.24-13.12.24	*Виконано*
7.	Оформлення третього розділу	13.12.24-14.12.24	*Виконано*
8.	Виконання завдання до підрозділу "Охорона праці"	08.12.24-09.11.24	*Виконано*
9.	Виконання завдання до підрозділу "Безпека в	10.12.24-12.12.24	*Виконано*
	надзвичайних ситуаціях"		
10.	Оформлення кваліфікаційної роботи	12.12.24-31.12.24	*Виконано*
11.	Нормоконтроль	14.12.24-15.12.24	*Виконано*
12.	Перевірка на плагіат	15.12.24	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.12.24	*Виконано*
14.	Захист кваліфікаційної роботи	23.12.2024	

Студент _____ Постолюк Т.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Липак Г.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Покращення процесів розробки додатків баз даних за допомогою впровадження практик CI/CD" // Кваліфікаційна робота освітнього рівня "Магістр" // Постолюк Тарас Миколайович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група САм-61 // Тернопіль, 2024 // с. – 62, рис. – 8, табл. – 2, джерел – 40.

Ключові слова: DevOps, розробка програмного забезпечення, безперервна доставка, безперервна інтеграція, розробка баз даних

Безперервна інтеграція та безперервна доставка (CI/CD) автоматизують процеси інтеграції програмного забезпечення, знижуючи обсяг повторюваної праці для інженерів. Хоча впровадження CI/CD сприяє підвищенню продуктивності, цей потенціал не використовується в повній мірі при розробці додатків для баз даних. У цій роботі проведено аналіз сучасного стану цієї області, зокрема розглянуті актуальні методи, основні програмні інструменти, виклики та умови, що стосуються програм, які працюють з базами даних.

Огляд літератури став основою для створення нового загального CI/CD конвеєра для розробки додатків, що взаємодіють із системами баз даних. Цей конвеєр був адаптований до трьох варіантів промислових розробок, де оцінювались переваги інтеграції та автоматизації процесу розгортання. Результати вимірювань чітко показали, що впровадження CI/CD дає значні переваги, зокрема зменшує кількість неуспішних розгортань, підвищує їх стабільність і дозволяє збільшити кількість релізів.

ANNOTATION

"Improving of Database Application Development Processes by Implementing CI/CD Practices" // Master's degree qualification paper // Postoliuk Taras Mykolayovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group САм-61 // Ternopil, 2024 // p. – 62, fig. – 8, tables – 2, references – 40.

Key words: DevOps, software engineering, continuous delivery, continuous integration, database development

Continuous Integration and Continuous Delivery (CI/CD) automate the software integration processes, reducing the amount of repetitive work for engineers. While the implementation of CI/CD contributes to increased productivity, this potential is not fully utilized in the development of database applications. This paper analyzes the current state of the field, focusing on current practices, key software tools, challenges, and conditions related to programs interacting with databases.

The literature review served as the foundation for creating a new general CI/CD pipeline for developing applications that interact with database systems. This pipeline was adapted to three industrial development scenarios, where the benefits of integration and deployment automation were evaluated. Measurement results clearly showed that implementing CI/CD provides significant advantages, including reducing the number of failed deployments, improving their stability, and increasing the number of releases.

ЗМІСТ

ВСТУП

Актуальність задачі.

Сучасні практики програмного забезпечення включають різноманітні методи, такі як гнучка розробка програмного забезпечення, мікросервіси, рефакторинг, тестування та неперервне розгортання. В останні роки безперервна інтеграція та безперервна доставка (CI/CD) стали стандартом у розробці програмного забезпечення. Зі збільшенням робочого навантаження повторювана робота з розгортання повинна бути автоматизована, щоб дозволити командам зосередитися на створенні цінності. Застосування CI/CD дозволяє розробникам швидко реагувати на зміну вимог користувачів і допомагає їм часто постачати перевірене програмне забезпечення без додаткової ручної роботи, що скорочує час виходу на ринок. Переваги впровадження CI/CD включають швидші та частіші випуски, автоматизоване безперервне тестування та гарантію якості, коротші цикли зворотного зв’язку, покращену надійність випуску та автоматизацію завдань, які раніше виконувалися вручну. Зі розширенням застосування технологій DevOps практики CI/CD, такі як контроль версій, автоматизація, CI, тестування та розгортання, стають все більш критичними.

Питаннями автоматизованого розгортання та програмної інженерії загалом займались також вчені в ТНТУ, зокрема, в роботах [2, 3, 4] досліджувались питання процесів CI/CD в гнучких технологіях розробки, ефективності процесів неперервного розгортання та їх вплив на якість створюваного продукту.

Мета роботи.

Метою роботи є дослідження процесів CS/CD для автоматизації розгортання програмного забезпечення з базами даних і розгортання самих баз даних.

Для досягнення мети потрібно вирішити наступні задачі:

1. Визначити міру застосування практики CI/CD в проектах зі створення ПЗ з базами даних.

2. Виявити основні проблеми, що виникають у застосуванні практик CI/CD до проектів програм баз даних.

3. Визначити переваги, що з'являються при використанні практики CI/CD для реальних проектів додатків баз даних

Об’єкт дослідження: процеси автоматизації розробки ПЗ з базами даних.

Предмет дослідження: автоматизоване розгортання ПЗ з базами даних.

Наукова новизна отриманих результатів. Хоча впровадження CI/CD підвищує ефективність розробки, воно все ще створює серйозні проблеми, пов’язані з існуванням даних, які розробляються незалежно від програми [1]. Тому в цій роботі проведено дослідження на предмет того, що зміни в програмах баз даних є технічною задачею, яку необхідно вирішити за допомогою автоматизації CI/CD. Якщо цього не зробити, це може призвести до серйозних вузьких місць у процесі випуску релізу.

Практичне значення отриманих результатів. Дана кваліфікаційна робота дозволить ефективно впроваджувати процеси CI/CD для розробки програм з базами даних з максимальною ефективністю.

Апробація результатів та особистий внесок здобувача. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції "Актуальні задачі сучасних технологій – 2024", яка проводилась у ТНТУ.

1 ОГЛЯД ЛІТЕРАТУРИ ЗА ТЕМОЮ РОБОТИ

Щоб оцінити стан застосування CI/CD в області розробки програм з базами даних, було проведено огляд літератури. Пошук джерел здійснювався у відкритих електронних бібліотеках (IEEE Xplore, ACM Digital Library, DBLP, Scopus і Web of Science), використовуючи терміни Database Application Development, CI і CD.

Таким чином огляд допоміг зрозуміти статус CI/CD у розробці додатків баз даних, включаючи програмні засоби, проблеми та організаційні передумови для впровадження CI/CD. Встановлено, що існує мало досліджень, які демонструють переваги впровадження конвеєрів CI та CD у великі проекти розробки додатків баз даних. Імовірно, що це пов’язано з тим, що для підтвердження таких переваг потрібно провести вимірювання під час розробки, інтеграції та розгортання, щоб перевірити, чи покращує автоматизація процес розробки.

Виходячи з того, що ми досліджуємо відкриту дослідницьку сферу, ми взялися за розробку та реалізацію конвеєра CI/CD, який враховує перспективу розробки додатків бази даних. У відкрити джерелах також було знайдено та проаналізовано практику практичної перевірки впровадження та аналізу конвеєру безперервної інтеграції у співпраці з трьома галузевими партнерами, які виконували проекти розробки додатків баз даних.

Три випадки промислового використання використовували базу даних Oracle і впровадили CI/CD для оптимізації розробки програм баз даних шляхом покращення процесів розгортання змін схеми бази даних і автоматизації повторюваних завдань. Перед цим конвеєри безперервної інтеграції не впроваджувались у досліджуваних проєктах. Розробники створили конвеєри інтеграції та доставки для всіх трьох випадків використання та виміряли продуктивність команди до та після впровадження конвеєрів. Було виміряно ефективність нової реалізації CI/CD як до, так і після впровадження.

Показники розраховуються для кожного розгортання, середовища, де встановлено зміни, або загалом. Вимірювання було проведено, проаналізовано

та оцінено. Очікується, що ці три варіанти використання підвищать незалежність всередині команди розробників щодо випуску та встановлення змін бази даних. Команди також сподівалися, що автоматизація зробить розгортання більш відтворюваними та підвищить загальну якість розробки та розгортання.

В роботі [5] описано загальний конвеєр CI/CD для програм баз даних і показано два конвеєри CI/CD, які покращують процес тестування та розгортання програмного забезпечення. Таким чином в результаті виконання дослідження досягнуто таких результатів:

– виконано огляд літератури, щоб пролити світло на використання методів CI/CD у розробці додатків баз даних;

– виявлено ще один, третій, варіант використання CI/CD, який інтегрує пов’язані з базами даних кроки розробки програми;

– надано додаткові деталі та глибший аналіз трьох промислових прикладів, що включають розгортання великих баз даних;

– проаналізовано результати кількісних і якісних оцінок конвеєрів на основі вимірювань і відгуків, зібраних у трьох випадках використання.

Конвеєр CI/CD, запропонований у [5], забезпечує еталонну архітектуру для тих, хто хоче застосувати CI/CD у розробці додатків баз даних. Три випадки використання застосовують еталонну архітектуру, як план для своїх реалізацій CI/CD, показують необхідні передумови, наводять приклади реалізацій, показують можливі налаштування ланцюга інструментів і кількісно оцінюють переваги CI/CD у розробці додатків бази даних.

Загалом мвиявлено значне зменшення кількості невдалих розгортань у більшості випадків (максимум 90% скорочення тестового середовища), що є чіткою ознакою покращення розгортання та якості коду. Також з'ясовано, що розробники виграють від меншого когнітивного навантаження.

1.1 Опис методу визначення продуктивності конвеєра CI/CD

Автоматизований CI/CD є основою сучасної розробки програмного забезпечення. CI передбачає автоматичне створення та тестування системи після внесення змін до вихідного коду. Це гарантує працездатність системи та забезпечує швидкий зворотній зв'язок команді розробників. Автоматизація гарантує роботу програми та її компонентів одним натисканням кнопки. Весь процес створення та розгортання потрібно автоматизувати, щоб ми могли перевірити його автоматично. У налаштуваннях бази даних це означає, що коли вихідний код бази даних зберігається так само, як і будь-який інший вихідний код системи, можна створити та інтегрувати код бази даних як частину процесу створення інтеграції програми. CI виконує збірку, яка включає компіляцію вихідного коду, створення програми, виконання модульних тестів і застосування конфігураційних файлів, інтеграцію, а також статичний аналіз коду.

Автоматизація збірки, автоматизоване тестування та надійне налаштування CI є передумовами для CD. CD автоматизує процес встановлення. Автоматизація збірки гарантує відтворюваність встановлення та незалежність від середовища розробки. CD дозволяє розробникам випускати програмне забезпечення надійно, швидко, високоякісно, з низьким рівнем ризику та ефективним способом. CD автоматизує всі необхідні кроки, поки програмне забезпечення не буде готове до встановлення та запуску у робочому середовищі. Реалізація CD називається конвеєром розгортання. Це гарантує, що програмне забезпечення знаходиться в придатному для випуску стані та завжди готове до автоматичного розгортання у виробництві.

Швидші зміни програмного забезпечення просуваються від розробки до запуску у виробництво; чим кращий відгук отримають розробники, тим швидше вони зможуть покращити програмне забезпечення. У цьому контексті використовується термін виконання робіт. Час виконання – це час між початком реалізації функції та її випуском у виробництво. Він розповідає нам, як постійно функції постачаються у виробництво. В ідеалі час виконання має бути якомога

коротшим, показуючи, що невеликі функції впроваджуються та постачаються безперервно без значних накладних витрат. Відношення відмов до всіх змін у розгортанні називається коефіцієнтом відмов змін.

Вимоги до швидкості роблять швидку збірку та інтеграцію програмного забезпечення дуже важливими. Складання поєднує програмне забезпечення, тестує та перевіряє, чи працює програма належним чином. Конвеєри CI автоматизують виробничі установки, тоді як конвеєри CD автоматично випускають програмне забезпечення у продакшн щоразу, коли є нова версія. Оскільки CI має надавати швидкий зворотний зв’язок розробникам, конвеєр CI/CD має бути якомога швидшим. У нашому аналізі CI/CD до та після впровадження в усіх трьох випадках використання ми використовуємо такі показники. Кількість функцій на розгортання. Мета полягає в тому, щоб зменшити розмір розгортання, щоби зменшити ризик, пов'язаний зі змінами. Чим менше розгортання, тим менший ризик.

Частота розгортання – скільки розгортань виконується в CI, невиробничих середовищах (non-production environments – NPE), також відомих як інтеграція, і виробничих середовищах на тиждень або місяць. Мета полягає в частому розгортанні, щоб зменшити ризик збоїв при розгортанні та отримати впевненість у мінімальних збоях, тому більша кількість розгортань вказує на те, що функції постачаються частіше. Це число залежить від проекту, і його не слід порівнювати між проектами, оскільки кожен проект має свій власний характер і динаміку.

Люди, здатні виконувати розгортання ПЗ на продакшн – кількість людей, які ефективно контролюють розмір і швидкість продуктивного розгортання.

Відсоток невдалих розгортань – показник надійності та стабільності розгортання. Чим надійнішим і стабільнішим є робочий процес, тим менше розгортань буде невдалими і меншим буде відсоток збоїв. В ідеалі це число близьке до нуля, що означає добре відлагоджений процес CI/CD, який виявляє помилки дуже рано. Цей відсоток було виміряно в усіх трьох випадках: тест, інтеграція та виробництво.

Відсоток автоматично перевіреного коду, пов’язаний із обсягом ручного тестування. Ручне тестування є антишаблоном у розробці. Це слід робити лише епізодично, оскільки це безпосередньо пов’язано з тим, скільки коду автоматично перевіряється. Чим менше коду перевіряється автоматично, тим більший обсяг ручного тестування. Чим більший обсяг ручного тестування, тим довший час тестування. Однак 100 % покриття коду зазвичай неможливе.

Ручне тестування на тиждень у відсотках від можливостей команди – кількість часу, який займає ручне тестування.

Час виконання – час між початком впровадження функції та її випуском у виробництво.

Час відновлення середовища – час, що минув для початку відновлення після невдалого розгортання на NPE або робочому місці. Він показує, скільки часу потрібно механізмам відновлення в проекті. Більше число показує, що знадобиться більше часу для відновлення, якщо розгортання залишає середовище в несправному стані. В ідеалі це число є якомога меншим, що зводить можливий простій системи до мінімуму.

1.2 Тестування бази даних

Ми починаємо з теми тестування в контексті бази даних, оскільки це невід’ємна частина конвеєрів CI/CD, потім обговорюємо ширшу проблему CI/CD у системах баз даних, перейдемо до програмних інструментів, що використовуються в CI/CD, і закінчимо обговоренням технічних та організаційних передумов використання CI/CD у цьому контексті.

Програма бази даних зазвичай складається з кількох рівнів, які потрібно перевірити, спочатку один за іншим, а потім як інтегровану систему. Стандартна архітектура програми бази даних показана на рисунку 1.1. Вона включає такі компоненти: прикладний рівень (верхній), рівень доступу до бази даних (API, середній) і рівень постійних даних (нижній).

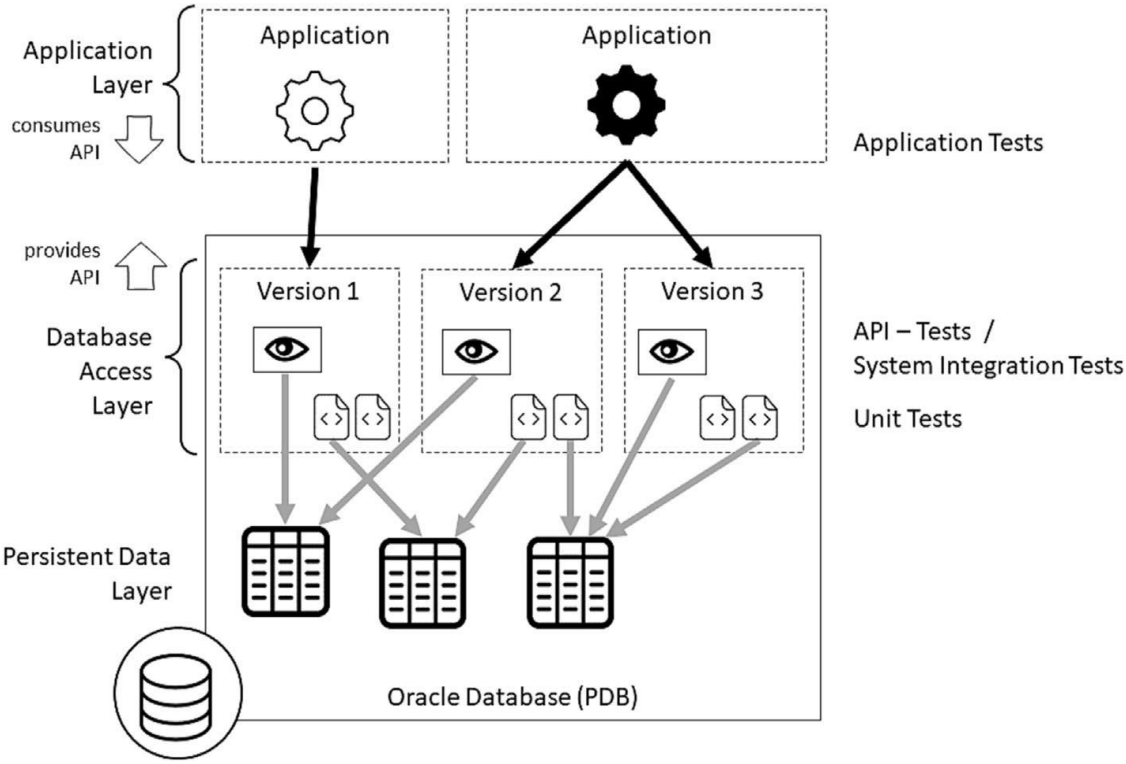

Рисунок 1.1 – Архітектура прикладної програми бази даних, що демонструє архітектуру рівня підключеної бази даних Oracle (PDB) і тести, необхідні для кожного рівня.

В ідеалі системи керування базами даних (СКБД), тобто нижня частина всієї програми, повинні підтримувати паралельне керування версіями схеми, реалізоване такими інструментами, як GIT, Mercurial або SVN. Однак керування версіями є складною проблемою для СУБД [6, 7].

У сучасній практиці версійний рівень доступу до даних знаходиться на рівні прикладного програмного інтерфейсу (API). API дозволяє розробникам опосередковано отримувати доступ до функцій бази даних і може підтримувати керування версіями. API покращують якість програмного забезпечення, надаючи простий багаторазовий інтерфейс для функціональність версійним способом, що зменшує навантаження на роботу з розробки. Постійні зміни є одним із двигунів еволюції API, що дозволяє API залишатися цінним і актуальним. Однією з головних цілей розвитку API є уникнення шкідливих змін, забезпечуючи функціональність.

Рівень доступу до бази даних із керуванням версіями (посередині рисунка 1.1) є передумовою для еволюції програми та рефакторингу існуючих таблиць бази даних. Рівень доступу до бази даних, який містить представлення та процедури, створює абстракцію, інкапсулюючи доступ і приховуючи фактичні об’єкти. Цей рівень абстракції також підтримує надання доступу для кожного об’єкта, що важливо для безпеки бази даних. Якщо API має версії, база даних може мати незалежний цикл випуску [7]. Однак для забезпечення безперервної роботи може знадобитися міграція системи бази даних без простоїв у деяких системах, наприклад у банківських програмах "бізнес-споживач". Відомі рішення цієї проблеми детально описані нижче.

Тестування запитів до бази даних є складнішим, ніж тестування в налаштуваннях без бази даних. Це пов’язано з тим, що кількість і типи входів і виходів можуть відрізнятися для кожного запиту, надісланого до бази даних, тоді як у програмі, яка не базується на базі даних, зазвичай є лише певний вхід і вихід, які потрібно перевірити для певного методу чи функції. Під час написання запитів до бази даних запит має повністю відповідати схемі, щоб працювати, а результат залежить від стану програми. Це означає, що тести повинні проходити за заздалегідь визначеним графіком, щоб отримати правильні результати.

Більшість сучасних баз даних пропонують процедурну мову, яка використовується для реалізації бізнес-логіки, що знаходиться всередині бази даних, забезпечуючи швидке маніпулювання даними, перетворення та обчислення. Цю логіку потрібно перевірити, коли вона зберігається в базі даних, у збережуваних процедурах, тригерах і функціях. Тестування обчислень, запрограмованих як функції або процедури в базі даних, належить до категорії логічного тестування. Тести логіки бази даних – це модульні тести, які мають передувати тестам API та інтеграції.

Традиційні тести API бази даних приховують базу даних за макетними об’єктами, щоб прискорити модульне тестування. Підхід до тестування API, який застосовується як до еволюції схеми, так і до рефакторингу бази даних, виконує оператори SQL select. Він перевіряє функціональність схеми після

еволюції, але не повертає дані, щоб забезпечити швидке виконання тесту. Ці оператори вибору розробляються одночасно з додатком і діють як договір між додатком і базою даних. Це відображає той самий принцип, який використовується в тестуванні контрактів, керованих споживачами [8], з тією різницею, що контракт між програмою та базою даних є операторами SQL, які не повертають дані.

Одним із способів тестування системної інтеграції є виконання запитів до тестованої бази даних. Запити можуть генеруватися автоматично або зберігатися як запити, які програма використовує для доступу до системи баз даних. Автоматично створені запити усувають необхідність вручну програмувати та підтримувати великі набори тестів, з недоліком, що запити будуть вибрані випадковим чином. Інший автоматично створений підхід до тестування системної інтеграції – тестування попарного покриття (pairwise coverage testing – PCT). PCT зменшує кількість перевірених комбінацій за допомогою методики попарного тестування. PCT – це метод тестування чорної скриньки, який забезпечує стовідсоткове охоплення тестом. Використовуючи репрезентативний вибірковий набір даних із двома значеннями та межами для кожного параметра, кількість тестів зменшується. Для роботи PCT потрібні правильні значення в тестових кейсах, тому в такому підході можна створити лише позитивні тестові кейси.

2ОСОБЛИВОСТІ CI/CD У ПРОГРАМАХ З БАЗАМИ ДАНИХ

2.1 Практики CI/CD у програмах баз даних

У той час як CI/CD є стандартним для програмного коду, код бази даних часто розгортається вручну. Це пов’язано з тим, що в програмах баз даних ми маємо справу зі станом, тобто даними. Ці дані часто потрібно перенести до нової схеми та, можливо, доведеться змінити, оскільки нові програмні модулі вимагають інших даних. Зміни даних у великій базі даних займають багато часу. Ось чому часто віддається перевага ручному розгортанню коду бази даних, щоб уникнути затримки. Однак це не ідеально з точки зору якості коду та ефективної розробки програмного забезпечення, оскільки відомо, що автоматизація завдань, пов’язаних із розробкою, покращує якість системи та економить ресурси з часом [9]. Отже, підвищення ефективності в розробці призводить до проблем, особливо тих, які пов’язані з існуванням даних, які потребують розробки.

Питання затримки в розгортанні коду бази даних уже деякий час було в центрі уваги. Наприклад, у [10] стверджують, що керування станом бази даних під час тестування є однією з головних причин тривалого тестування, а тестування додатків бази даних суттєво відрізняється від тестування додатків, де БД не задіяна. Вони також досліджували розпаралелювання тестів і розробили рішення для динамічного планування тестів, щоб скоротити час, необхідний для розгортання. Інший варіант, генерація даних, був досліджений у [11], де автори створили генератор баз даних і інструмент тестування, який генерує значущі тестові бази даних, виконує ефективне тестування додатків баз даних і запускає тести паралельно. Потім вони розширили свою попередню роботу та продемонстрували структуру для ефективного тестування додатків бази даних.

У роботі [12] автори працювали над створенням додатків і генеруванням рівня трансляції, щоб вирішити проблему еволюції схеми та додатків, щоб уникнути руйнівних змін. Інший внесок у розвиток схеми та інструментів для безперервного розгортання показав, як перейти з однієї схеми бази даних на нову

та переконатися, що клієнти бази даних залишаються функціональними. Цю роботу розширили автори, показавши, що база даних зі змішаним станом підтримує плавний перехід між схемами під час еволюції. Пов’язану проблему технічних і соціальних проблем, які спостерігаються в CI/CD, було досліджено та вирішено за допомогою стратегій пом’якшення впливу на бізнес без налаштування CI/CD [13].

Інші рішення для керування даними під час розгортання коду зосереджені на уникненні критичних змін у безперервному розгортанні. Серед них нещодавно представлені шари абстракції та відображення [14]. Автори пропонують механізм відмінності схем, який порівнює об’єкти між двома базами даних або двома схемами, розпізнає можливі несправні зміни та виправляє несправні зміни шляхом запису з’єднань таблиці для перенесеної схеми. Пов’язаним рішенням у цій галузі є шаблон толерантного зчитувача, який є вказівкою щодо максимально толерантного проектування під час програмного зчитування даних зі служби, щоб уникнути руйнівних змін [15**Error! Reference source not found.**].

Серед робіт, присвячених тестуванню, ми відзначимо ряд, які досліджували тестування коду бази даних у проектах розробки системи. Вони виявили, що код бази даних часто погано перевіряється, і в цій галузі відчувається брак інструкцій. Проблеми, з якими вони зіткнулися, автори класифікують на практичні та концептуальні. На концептуальному рівні вони відзначають наступне: методи тестування, проблеми перевірки тестів, проблеми підтримуваності/тестування та методологічні підходи. На технічному (практичному) рівні вони перераховують питання, пов’язані з керуванням БД, підключенням до БД, заповненням БД перед тестуванням, вибором фреймворків, конфігурацією, моделюванням і паралелізацією.

Тема еволюції схеми також була досліджена. Наприклад, у [16] розглядали підходи, що підтримуються інструментами, моделювання та аналіз впливу змін. Інше дослідження напівавтоматизованої еволюції схем описує 75% економії часу при автоматизації створення коду еволюції бази даних [17]. Оскільки

розгортання може призвести до критичних змін, перспектива надійності є дуже актуальною. Попередні дослідження показали брак інструментів, які полегшують і перевіряють автоматизацію інтеграції та доставки додатків бази даних. Однак нещодавно у роботі [18] автори зосередилися на розробці надійності сайту, включаючи управління процесами розгортання, і дали достатню кількість порад щодо різних аспектів управління інфраструктурою, що мають відношення до контексту застосування бази даних.

Крім академічних досліджень, які обговорювалися досі, цікаво відзначити ряд промислових звітів від баз даних і великих постачальників додатків, які показують, що конвеєри CI/CD широко використовуються у великих організаціях, які покладаються на технології баз даних або створюють їх. Серед них [19], де розробка інтегрує тести продуктивності в передкоммітну частину конвеєра CI та має справу з довгостроковими тестами та сценаріями, які не можуть бути частиною попереднього тестування. Це мінімізує кількість ручної роботи для контролю інфраструктури CI та виявлення аномалій продуктивності та звітування про них.

Практики розробки MongoDB базуються на схожих принципах [20]. MongoDB запускає повністю автоматизовані тести продуктивності системи в середовищі CI. Автоматизація охоплює надання та розгортання великих кластерів, тестування, налаштування повторюваних результатів, а також збір і аналіз даних. Автоматизація вимірювань призводить до швидшого вдосконалення та підвищення якості, підвищує продуктивність інженерів-розробників і забезпечує більш продуктивний продукт.

Необхідність версій коду є важливим припущенням, що лежить в основі CI/CD. У той час, як сучасні конвеєри CI/CD дотримуються принципу конвеєра як коду, де конвеєр CI/CD генерується з коду, що зберігається в системі керування версіями (version control system – VCS), керування версіями змін бази даних не є стандартизованим процесом, прийнятим промисловістю. Ще одним аспектом, пов’язаним з надійністю, є розробки в рамках DataOps, які впроваджують аспект автоматизації в галузі науки про дані та машинного

навчання шляхом автоматизації інтеграції та доставки нових моделей у виробництво. Принципи DataOps подібні до DevOps і включають автоматизацію, усе як код, відтворюваність завдань, вбудовану якість і короткий час циклу [21].

Незважаючи на зосередженість на оцінці застосування різних підходів до розробки програмного забезпечення CI/CD і DevOps, не було знайдено недавніх робіт, які б описували, як реалізувати CI/CD і оцінювати застосування практик CI/CD для додатків баз даних. Крім того, в літературі бракує емпіричних досліджень того, як розробники перевіряють код доступу до бази даних на практиці.

2.2 Проблеми впровадження CI/CD у програмах баз даних

Часті зміни додатків баз даних слід вирішувати за допомогою автоматизації CI/CD, оскільки без автоматизації ми бачимо вузькі місця в процесі релізу. Автоматизована еволюція схеми бази даних є нетривіальною, і CI/CD для додатків реляційної бази даних все ще рідко застосовується; однак це залишається одним із найскладніших аспектів розробки баз даних [22]. Дизайн бази даних, тестування, якість даних і еволюція схеми є передумовами для успішного впровадження CI/CD. Однак наразі в більшості випадків автоматизована лише частина міграції схеми, а інші практики CI/CD, такі як автоматичне тестування чи статичний аналіз коду, рідко включаються. Під час тестування додатків баз даних однією з проблем є час, необхідний для надання даних для тесту [23].

У додатках баз даних виклики впровадження CI/CD складніші, ніж у додатках без постійного стану. У 2015 році лише 43% проектів розробки баз даних мали робочий процес розробки бази даних, автоматизований за допомогою CI/CD [24], і наші випадки промислового використання показують, що повна автоматизація все ще не є стандартною практикою. Відсутність адаптації може бути спричинена організаційними проблемами, наприклад, організаціями, які очікують гнучких методів від своїх команд розробників, але

не створюють необхідних організаційних структур, довірчого середовища та перекладання відповідальності на команди.

Проблеми в автоматизованому тестуванні додатків бази даних включають роботу з обробкою бази даних, включаючи розгортання змін і налаштування бази даних. Інженери-програмісти повідомили, що вони вважають автоматизацію тестування бази даних, тестове покриття та обробку еволюції схеми дуже складними під час розробки, особливо тому, що найкращі практики та рекомендації щодо тестування баз даних відсутні. Зокрема, бази даних потребують спеціальних підходів до тестування. Тестування бази даних має справу з двома аспектами: перевіркою коду бази даних і даних бази даних в одному тестовому прогоні. Тести коду бази даних перевіряють функціональність функцій і процедур у базі даних за допомогою модульних тестів, тоді як перевірки якості даних підтверджують правильність даних після міграцій.

Проблеми тестування баз даних і передовий досвід включають як концептуальні, так і технічні проблеми [25]. Одним із способів пом’якшення проблем є використання розробки на основі магістралі, яка вирішує проблеми інтеграції великих комітів, конфліктів злиття, несправних збірок, блокування роботи, тривалих розгалужень і повільного затвердження інтеграції [26]. У той же час, оскільки магістральна розробка не допускає розгалужень, вона вимагає впровадження прихованих змін, необхідних для впровадження більш масштабних змін. Крім того, з великою базою коду та командами, розділеними на кілька підгруп, все одно рекомендується розробляти основну вітку коду в спільній базі коду, щоб уникнути труднощів інтеграції та пошуку причин невдач, запровадженого розгалуженням. Іншою проблемою в еволюції системи баз даних є необхідність підтримки зміни схеми та кількох версій рівня доступу до даних у програмі бази даних. Якщо програмне забезпечення бази даних не надає таких функцій, як перевизначення на основі випуску Oracle (EBR), цю роботу потрібно виконати вручну [27].

Подібно до інших прикладних областей, в еволюції схеми бази даних кардинальні зміни можуть відбуватися під час безперервного процесу розробки

та вдосконалення програм. В еволюції схеми бази даних критична зміна відбувається, коли змінюється сигнатура або тип повернення API. У розробці бази даних API визначається рівнем доступу до бази даних, до якого мають доступ споживачі служби бази даних. Коли відбуваються зміни на цьому рівні, програма-споживач також має змінитися, якщо цей рівень не має версій. Рівень доступу без версії поєднує цикл випуску програми безпосередньо з циклом випуску бази даних і робить неможливим окремі розгортання. Наприклад, у [14] автори представляють стратегії вирішення цієї проблеми, однак їхній підхід до виправлення зламаного коду непридатний для промислового використання, де ми очікуємо, що програмне забезпечення працюватиме правильно 100% часу.

Зміни схеми бази даних можна розглядати як проблему проектування системи, яка заважає прийняти CD. Загальним рішенням є використання розгалуження за допомогою абстракції (Branching by Abstraction – BbA), яке вносить приховані зміни, викликаючи лише невеликі зміни в розробці на основі магістралі. BbA можна використовувати для впровадження значних змін, не порушуючи інші роботи з розробки. Зміни впроваджуються за шаром абстракції, який служить межею між існуючим кодом і кодом, що змінюється. Таким чином можна вносити великі зміни в базу даних, не створюючи проблем для інших розробників або користувачів додатків [28].

BbA включає наступні кроки, показані на рисунку 1.2.

1. Реалізація рівня абстракції над компонентом, який потрібно замінити.
2. Рефакторинг залежних компонентів для використання рівня абстракції замість компонента, який потрібно замінити.
3. Впровадження однієї або кількох нових версій компонента. Рівень абстракції переспрямовує на старий або новий компонент. Таким чином, обидві версії можуть співіснувати.
4. Коли все реалізовано в новому компоненті, видаліть компонент, який потрібно замінити.

5. За потреби можна повторити два попередні кроки та застосувати зміни.

6. Після завершення шар абстракції можна видалити, але це не обов’язково.

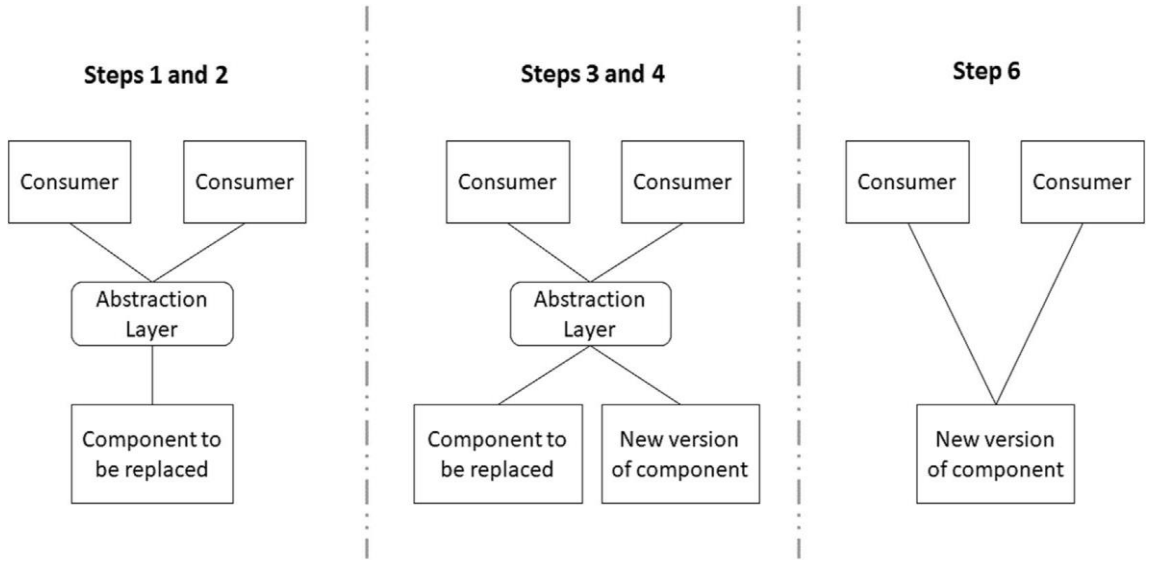

Рисунок 2.1 – Розгалуження за допомогою абстракції

Тепер звернемо увагу на проблеми налаштування тестування бази даних. Перш ніж міграцію схеми бази даних можна буде виконати під час CI, СКБД має бути доступною для запуску міграції. Тестування з базою даних дає розробнику швидкий відгук, чи працюють зміни чи ні. Наступні механізми можна використовувати для інтеграції змін і виконання побудови бази даних під час CI.

База даних у зображенні Docker – під час CI можна запустити контейнер докерів, який містить певний програмний стек для виконання інтеграції в ізоляції. Контейнерне виконання завжди пропонує чисте середовище без будь-яких змін конфігурації. Oracle пропонує експрес-версію бази даних у образі докера для підтримки виконання в контейнерах незалежно від операційної системи.

Стабільна база даних, що містить версію програмного забезпечення. Налаштування інфраструктури бази даних для CI/CD є проблемою, з якою стикаються розробники, коли вони хочуть реалізувати базу даних CI/CD.

Стабільна база даних, яка інтегрує всі зміни, але зберігає останній стан програмного забезпечення інтеграції, може зменшити складність ініціалізації інфраструктури та скоротити час виконання CI.

Стабільна порожня база даних для CI. Ще один підхід, щоб уникнути надання бази даних до CI, полягає у використанні порожньої бази даних, де попередня версія програмного забезпечення встановлюється перед виконанням міграції. База даних очищається після завершення CI, щоб бути готовою до наступного запуску інтеграції.

Клонована змінна база даних (pluggable database – PDB) – у базі даних Oracle можна швидко клонувати існуючу, так звану PDB, із шаблонної бази даних. Шаблон може містити стан програми для виконання конвеєра CI. Після успішного розгортання шаблон можна оновити до останнього стану.

Нова PDB – PDB можна легко створити в мультитенантній архітектурі Oracle, яка є архітектурою бази даних за замовчуванням. З новою PDB конвеєр CI можна запускати з чистою базою даних.

Налаштування бази даних часто є основною причиною тривалого часу виконання CI. Тому автори [25] пропонують використовувати або бази даних у пам’яті, або стабільні, уже готові бази даних для запуску CI. Ці середовища очищаються після тестування. Проблема з базами даних у пам’яті полягає в тому, що вони зазвичай не пропонують такої ж функціональності, як вихідна база даних, і створюють більший ризик для процесу CI.

Усі виклики, про які ми повідомляли, ускладнюють розробникам додатків баз даних впровадження CI/CD. Зокрема, відсутність еталонних архітектур вимагає багато концептуальної роботи та оцінки та впровадження інструментів. На основі знань, які ми зібрали та узагальнили в цьому розділі, ми розробили загальний конвеєр CI, показаний на рисунку 1.3.

Конвеєр, який ми пропонуємо, включає всі необхідні кроки, знайдені в літературі. У цьому конвеєрі кожна нова фіксація запускає частину CI, яка включає статичний аналіз коду, виконання сценаріїв міграції бази даних,

завдання після міграції, виконання тесту, встановлення номера випуску, упаковку артефакту та розміщення його в сховищі артефактів.

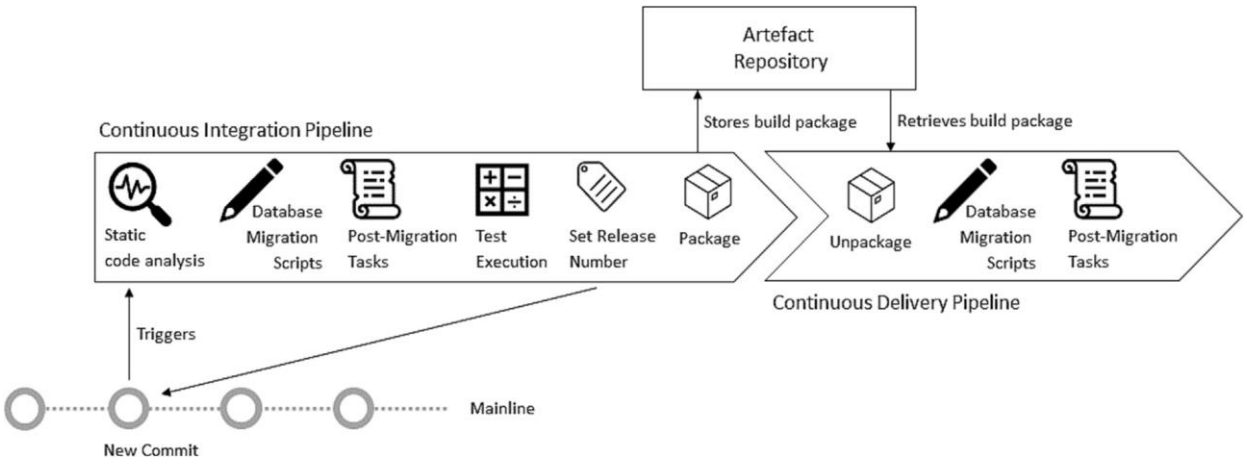

Рисунок 2.2 – Загальний конвеєр, що включає всі основні етапи безперервної інтеграції та доставки додатків бази даних

Потім може розпочатися частина CD з розпакуванням артефактів, міграцією бази даних і завданнями після міграції.

2.3 Програмні засоби міграції та тестування баз даних

У розробці додатків бази даних інструменти міграції бази даних забезпечують усі функції, необхідні для автоматизації та відстеження розгортання змін схеми бази даних без необхідності написання спеціальних сценаріїв. Результати опитування з вільних джерел показало, що понад для 100 розробників (див. рис. 1.4) найпопулярнішими інструментами міграції баз даних у спільноті Oracle були Flyway і Liquibase [18].

Багато реляційних баз даних підтримують функції, процедури та пакети всередині бази даних, написані мовою структурованих запитів (SQL). У деяких з них, наприклад Oracle, автоматизоване тестування всередині бази даних можливе за допомогою модульного тестування. Найпопулярнішою структурою модульного тестування для Oracle є utplsql (utplsql.org). Наразі це єдиний зрілий інструмент модульного тестування на ринку для Oracle, що працює в базі даних. Інші інструменти, як-от dbUnit (dbunit.org), розширення jUnit (junit.org), також

підтримують модульне тестування, але для запуску потрібна додаткова програма Java. Окрім модульних тестів, вихідний код бази даних слід аналізувати за допомогою статичного аналізу коду. Статичний аналіз коду перевіряє синтаксичну правильність коду та повідомляє про можливі помилки.

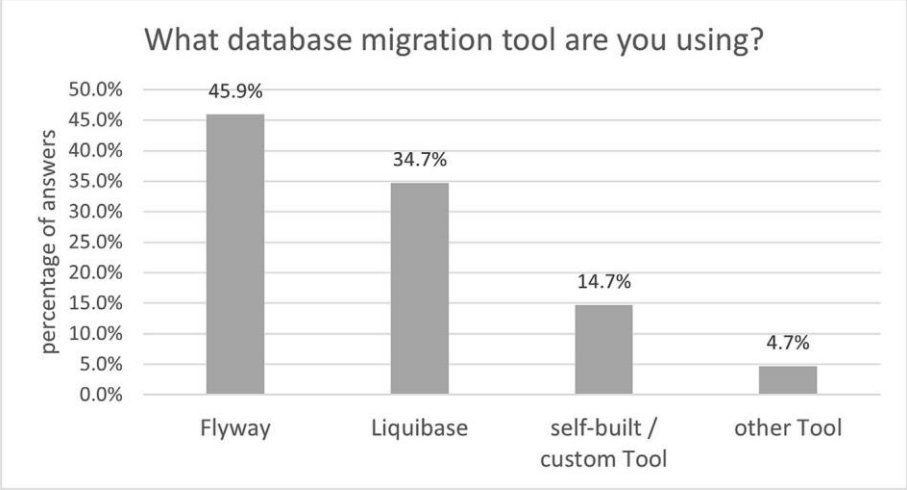

Рисунок 2.3 – Результати опитування інструменту міграції Oracle, де показано, що більшість розробників використовують Flyway і Liquibase

Інструменти статичного аналізу коду SQL поділяються на дві категорії: засоби перевірки синтаксису та засоби перевірки синтаксису й уразливостей. Засоби перевірки синтаксису (інструменти linting) включають SQLint (github.com/purcell / sqlint), який перевіряє код на відповідність стандарту ANSI за допомогою синтаксичного аналізатора Postgres SQL, SQL Fluff (sqlfluff.com), який використовує власний аналізатор для перевірки на наявність помилок і дефектів форматування, а також CODECOP (github.com/ Trivadis / plsql -cop-sqldev), який перевіряє Код SQL і PL/SQL для виявлення порушень інструкцій із кодування Trivadis (trivadis.github.io/ plsql -and- sql -coding-guidelines) і надає кілька показників коду. Засоби перевірки синтаксису та вразливості включають більш складні інструменти, такі як SQL Check (analysis- tools.dev /tool/ sqlcheck), який перевіряє код на наявність поширених антишаблонів і вразливостей SQL. Z PL/SQL Analyzer CLI (felipezorzo.com.br/zpa) перевіряє код SQL і PL/SQL на

наявність помилок і дублювання коду, а також обчислює такі показники, як складність і розмір коду. Окрім безкоштовних інструментів CLI, існують серверні інсталяції інструментів статичного аналізу коду, які перевіряють, чи відповідає код інструкціям із кодування чи має він вразливості, а також обчислюють показники коду. Популярними інсталяціями сервера є SonarSource (sonarsource.com) і Codacy (codacy.com).

Решта інструментів, як-от репозиторій контролю версій, репозиторій артефактів і сервер CI, не відрізняються для CI/CD бази даних і для CI/CD програми. Можна використовувати такі популярні репозиторії контролю версій, як GitHub (github.com), GitLab (gitlab.com) або Bitbucket (bitbucket.org). Усе вищезазначене також забезпечує функціональність сервера CI/CD. Автономні сервери CI/CD, як-от TeamCity (jetbrains.com/ teamcity) або Jenkins (jenkins.io), відокремлюють CI/CD від керування початковим кодом, якщо це потрібно. Репозиторії артефактів, такі як Artifactory (jfrog.com/ artifactory) або Nexus Repository (sonatype.com/products/nexus-repository), підтримують зберігання та отримання артефактів розгортання.

Ми також відзначаємо деякі людські фактори, які ускладнюють впровадження CI/CD у контексті бази даних. Той факт, що програмісти звикли зберігати код бази даних у базі даних як визначення таблиць, процедури чи функції, створює залежність від існуючої бази даних і ускладнює програмісту звикнути до розробки на основі файлів за допомогою VCS. Щоб експортувати метадані об’єкта з бази даних у систему контролю версій, розробники мають використовувати інструменти. IDE для розробки баз даних надають низку функцій експорту метаданих за допомогою майстрів або меню навігації об’єктів [29, 30]. Командний рядок SQL Developer (SQLcl) надає інтерфейс командного рядка до бази даних Oracle. Це дозволяє розробникам писати сценарії експорту, які вони можуть зберігати в репозиторії VCS. Такі сценарії можуть експортувати оператори мови визначення даних (DDL), які генерують таблиці та інші конструкції бази даних і відокремлюють SQL від функціональності візуальної IDE. Таким чином, команда розробників відповідає стандартам експорту

незалежно від налаштувань IDE. Для спеціальних вимог щодо експорту пакет PL/SQL під назвою DBMS_METADATA надає доступ до словника даних бази даних, що містить усю інформацію визначення об’єкта. Цей пакет також можна викликати з SQLcl і використовувати для отримання визначень об’єктів бази даних [30].

2.4 Технічні та організаційні передумови

Перш ніж групи розробників бази даних зможуть почати визначати конвеєр CI/CD, необхідно виконати кілька архітектурних, технічних і організаційних умов, які для команди розробників можуть бути серйозним викликом.

1. Автоматизація вимагає, щоб усі компоненти програми зберігалися в центральній системі керування версіями, включаючи всі конфігурації [18].
2. Автоматизація також передбачає наявність інструмента міграції бази даних і визначення структури каталогу керування версіями.
3. Має бути узгоджено попередньо визначений робочий процес розробки [5].
4. Необхідно визначити та виконати автоматизовані модульні та інтеграційні тести.
5. Має бути доступний статичний аналіз коду, який автоматично перевіряє, чи дотримуються стандарти кодування [31].
6. Архітектура системи бази даних і додатків-споживачів повинна бути побудована таким чином, щоб дозволити відокремленим випускам додатка бази даних не створювати простоїв. Було показано, що архітектури з високим ступенем зв’язку створюють серйозні проблеми під час впровадження автоматизації та CI/CD [32]. Для автоматизації потрібен рівень доступу до даних з версіями, який відокремлює програму від бази даних. Версійний рівень доступу до даних може відокремити програму від бази даних, а відокремлення рівнів від керування версіями запобігає руйнівним змінам доступу до програми.

ЗПОБУДОВА CI/CD-КОНВЕЄРУ З ДОДАТКАМИ БАЗ ДАНИХ

Зараз ми представляємо дизайн і налаштування інфраструктури конвеєра CI/CD, який об’єднує кроки, пов’язані з базою даних. Вони базуються на літературі, яку ми розглянули вище і на найкращих практиках розробки програмного забезпечення [33].

3.1 Проектування конвеєру

На рисунку 3.1 показано схему конвеєра, включаючи розробника, інфраструктуру та кроки конвеєра. Інфраструктура складається з сервера CI, системи контролю версій (VSC), інструменту статичного аналізу коду, бази даних CI, сховища артефактів і цільової бази даних. Розробник випускає код у VCS шляхом виконання коміту. Це запускає конвеєр, який складається з трьох частин: інтеграція, випуск і розгортання. Конвеєр виконує CI/CD у кілька етапів. Інтеграція складається з кроків 1–6.

На кроці 1, перевірці коду, конвеєр отримує зміни з VCS.

На кроці 2, статичний аналіз коду, код аналізується інструментом статичного аналізу коду, щоб перевірити, чи відповідає він інструкціям і умовам кодування.

На кроці 3, резервне копіювання, конвеєр створює резервну копію бази даних або точку відновлення.

На кроці 4, розгортання коду SQL, сценарії міграції схеми бази даних PL/SQL, закріплені в репозиторії, розгортаються в базі даних CI за допомогою інструмента міграції бази даних.

На кроці 5, Unit Tests, конвеєр запускає модульні тести коду SQL у базі даних CI. Якщо модульні тести пройшли успішно, на кроці 6 Системні тести запускаються системні тести для нової версії бази даних CI.

На цьому частина інтеграції закінчена. Наступна частина, випуск, складається з двох частин: на кроці 7, Set Release Number, номер випуску

встановлюється у VCS, а на кроці 8, Push Artifact, змінений артефакт надсилається до сховища артефактів. На цьому реліз завершено.

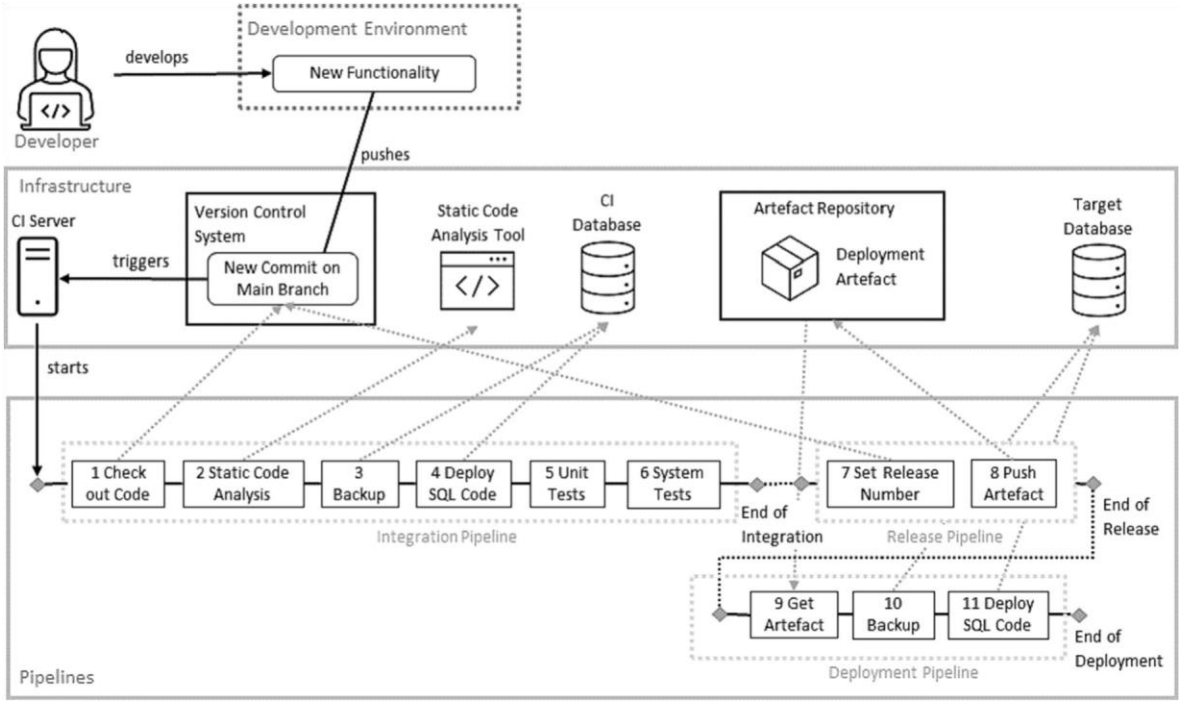

Рисунок 3.1 – Робочий процес розробки з використанням конвеєра безперервної інтеграції (CI) і розгортання (CD)

Конвеєр починає розгортання, яке складається з трьох кроків.

На кроці 9 «Отримати артефакт» конвеєр отримує артефакт, який можна розгорнути, зі сховища артефактів, на кроці 10 «Резервне копіювання» створює резервну копію цільової бази даних, а на кроці 11 «Розгорнути код SQL» розгортає цільову базу даних. Це закриває фазу розгортання, і весь конвеєр завершено.

3.2 Обґрунтування архітектури конвеєру

Тепер ми обговоримо причини, що стоять за структурою конвеєру. Основна мета проекту полягає в тому, щоб конвеєр вийшов з ладу раніше. Ці кроки мають виконуватися швидко та надавати швидкий зворотний зв’язок розробнику на ранніх етапах розробки; тривалі тести можуть бути виконані пізніше. Очевидно, що наступні функції допомагають покращити

продуктивність доставки програмного забезпечення: контроль версій для всіх виробничих артефактів, автоматизація змін бази даних, реалізація CI, магістральні методи розробки, реалізація автоматизації тестування, підтримка керування тестовими даними, реалізація CD, створення слабозв'язаної архітектури, збір і впровадження відгуків клієнтів, робота невеликими партіями та завчасна перевірка стану системи.

Це відображається в інфраструктурі та самому конвеєрі за допомогою таких функцій: VCS, сховище артефактів, модульні тести, автоматичне резервне копіювання та відновлення, а також автоматизація конвеєру.

Важливість використання VCS обговорюється в попередніх дослідженнях. VCS має містити все необхідне для створення програми. Наступні елементи бази даних повинні бути частиною репозиторію контролю версій [18]: міграції об'єктів бази даних, тригери, процедури та функції, представлення, конфігурації, сценарії очищення даних і зразки тестових наборів даних, які включають метадані та операційні дані та доступ до великої кількості набір даних для тестування продуктивності.

Статичний аналіз коду має бути частиною основного робочого процесу розробника, використовуватися як контроль якості на ранній стадії конвеєра CI та інтегруватися з IDE розробника, що виконується на кроці 5 (див. рис. 3.1).

Автоматизовані конвеєри вимагають повністю автоматизованих механізмів відкату, коли розгортання не вдається. Автоматичний відкат підвищує впевненість розробників і дозволяє їм частіше вносити зміни в робочу версію. Відсутність автоматизованих механізмів може змусити команду розгортати рідше, оскільки їй потрібні ресурси, якщо під час розгортання не вдається виправити цільову версію бази даних вручну. Відкат вмикається шляхом створення резервних копій і визначення точки відновлення в кроках 3 і 10 (див. рис. 3.1).

У великих проектах розробки додатків баз даних неможливо створити цілу програму бази даних з нуля з порожньою базою даних. Неможливо надати розробникам швидкий зворотний зв'язок, якщо базу даних потрібно створювати

під час кожної інтеграції. Через ці часові обмеження використовується стабільне середовище CI, яке вже містить тестові дані (називається базою даних CI, частиною інфраструктури, яку ми визначаємо). У разі невдачі зміни автоматично повертаються до попередньо встановленої точки відновлення, забезпечуючи автоматичний механізм відновлення.

Модульні тести виконуються як крок 5 (див. рис. 3.1). Оскільки модульні тести займають менше часу, ніж системні тести, вони запускаються першими, щоб забезпечити швидкий зворотний зв’язок і гарантувати, що збій виникне на ранній стадії, відповідно до принципу «швидкий і частий збій» [9]. Коли модульні тести завершуються успішно, слідують системні тести (крок 6).

В ідеальному сценарії розробник повинен мати можливість виконати всі кроки конвеєра CI локально, перш ніж він внесе зміни в спільне віддалене сховище VCS. Таким чином, перший тест регресії та інсталяції вже виконується локально до запуску віддаленого конвеєра CI.

3.3 Інфраструктура конвеєра CI/CD для програм баз даних

Схема інфраструктури показана на рисунку 3.2. У верхній лівій частині зображено розробника. Середовище розвитку ідеально є персоналізованим. Спільне середовище можливе, якщо програма містить достатньо різних об’єктів, щоб кілька груп розробників працювали над функціонально розділеними областями. Для невеликих додатків краще використовувати окремі середовища. Однак для досягнення найвищої якості середовище розробки має бути якомога схоже на виробниче середовище, щоб виявити помилки на ранній стадії та уникнути дрейфу версій програмного забезпечення бази даних.

Коли реалізація в середовищі розробки завершена, розробник додає DDL об’єктів бази даних до локальної VCS. Локальні сценарії експорту можна використовувати для експорту файлів DDL у VCS, уникаючи копіювання та вставлення. Сценарії експорту експортують DDL до локального репозиторію у вказаному форматі з усією необхідною інформацією. Потім розробник може

відправити їх на віддалений VCS. Розробник використовує або IDE, яка вже надає клієнт VCS для перевірки змін у системі керування версіями, або клієнт VCS інсталюється окремо на робочій станції розробника.

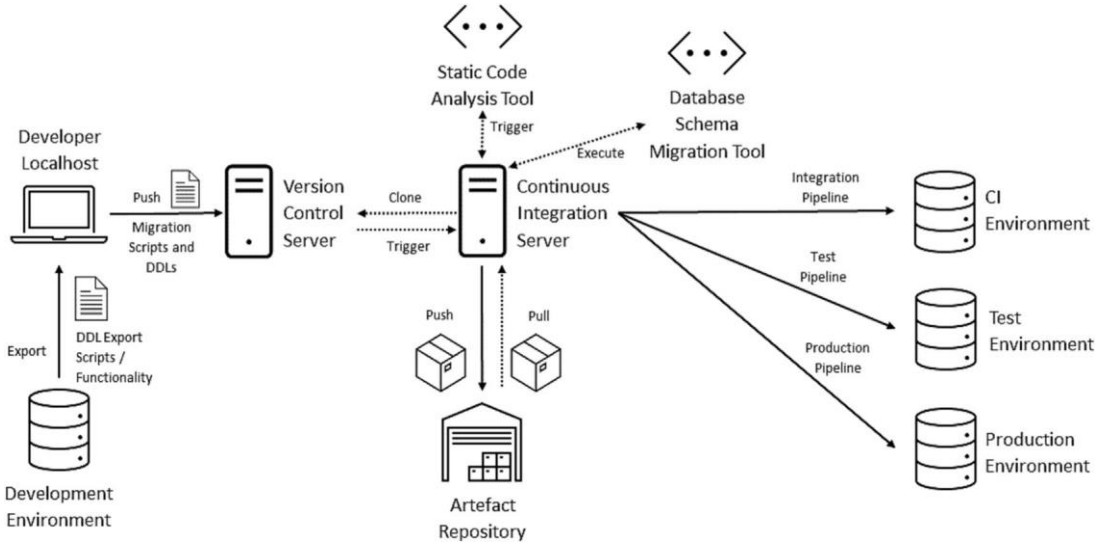

Рисунок 3.2 – Налаштування інфраструктури безперервної інтеграції (CI)/безперервної доставки (CD).

Тепер ми опишемо сервер CI, який потрібно інтегрувати з VCS, щоб запускати конвеєри, коли вносяться зміни в контроль версій. Конвеєрам CI потрібні дозволи на читання в сховищі вихідного коду, щоб перевірити код, створити артефакти розгортання та дозволи на запис, щоб позначати коміти номерами випусків. Дозволи на отримання та надсилання від сервера CI до сховища артефактів необхідні, щоб надсилати артефакт у сховище артефактів після того, як його було створено з нещодавно доданих змін VCS. Конвеєри CI надсилатимуть щойно створені артефакти розгортання до сховища артефактів. Конвеєри CD витягуватимуть артефакти зі сховища артефактів, щоб розгорнути їх у цільовому середовищі. Таким чином для конвеєрів розгортання гарантується принцип найменших привілеїв. Для сервера CI мають бути доступні як інструмент аналізу статичного коду, так і інструмент міграції схеми бази даних. Ці інструменти можна встановити на CI/CD як інструменти командного рядка безпосередньо, відкрити в контейнерному середовищі або бути доступними як інсталяція на сервері з API. Сервер CI повинен мати доступ до трьох цільових

середовищ, щоб виконувати розгортання коду та запускати тести. Потрібен користувач розгортання, який може інсталювати всі схеми цільової бази даних.

3.4 Випадки використання конвеєра CI/CD для програм баз даних

У цьому розділі ми представляємо три промислові приклади, пов'язані з великими проектами розробки додатків баз даних Oracle та згадані у джерелах, вказаних в попередніх розділах роботи. У всіх трьох проектах логіка присутня як збережувані процедури та функції PL/SQL у базі даних. Перший проект – розробка сховища даних (DWH) у страховій компанії; ми називаємо це випадком використання 1 (DWH). Другий випадок стосується розробки бази даних для бек-енду, і ми називаємо його варіантом використання 2 (BE). Третє дослідження – це розширена база даних у середовищі роздрібної торгівлі, яка позначена як випадок використання 3 (RET).

Щоб отримати уявлення про практику розробки, ми провели 90-хвилинні інтерв'ю з двома розробниками з кожного сценарію використання до та після впровадження CI/CD, загалом дванадцять інтерв'ю.

Таблиця 3.1 – Запитання до та після впровадження конвеєра CI/CD

ID	Питання щодо робочого процесу
Q1	Як виглядає ваш процес розгортання?
Q2	Які загальні проблеми?
Q3	Що добре працює?
Q4	Скільки часу ви витрачаєте на ручне тестування функцій?
Q5	Скільки часу вам потрібно для незапланованої роботи: підтримка, виправлення?
Q6	Як переконатися, що міграція бази даних успішна?
Q7	Скільки людей можна розгорнути?
Q8	Якби ви могли покращити робочий процес, як би ви це зробили?

Ми запитали про їхні погляди на робочий процес розробки бази даних і проблеми, які вони бачили. Опитані могли запропонувати можливі оптимізації.

Питання, які ми поставили, наведені в таблиці 3.1. Відповіді на ці запитання допомогли нам зрозуміти робочий процес розробки та згодом оцінити вдосконалення після впровадження конвеєра CI/CD.

3.5 Варіант використання сховища даних (DWH)

DWH – це проект розробки сховища даних у великій страховій компанії з 20 розробниками, які щотижня випускають зміни у виробництво, використовуючи попередньо визначені вікна розгортання для середовищ тестування та інтеграції. Сховище даних складається з приблизно 3000 таблиць, які заповнюються за допомогою збережуваних процедур, використовуючи близько 10 ТБ пам’яті.

Крім статичного середовища розробки, розробники мають кілька статичних середовищ, як-от середовище тестування продуктивності, яке є частиною процесу випуску. Робочий процес розгортання до автоматизації показано на рисунку 3.3.

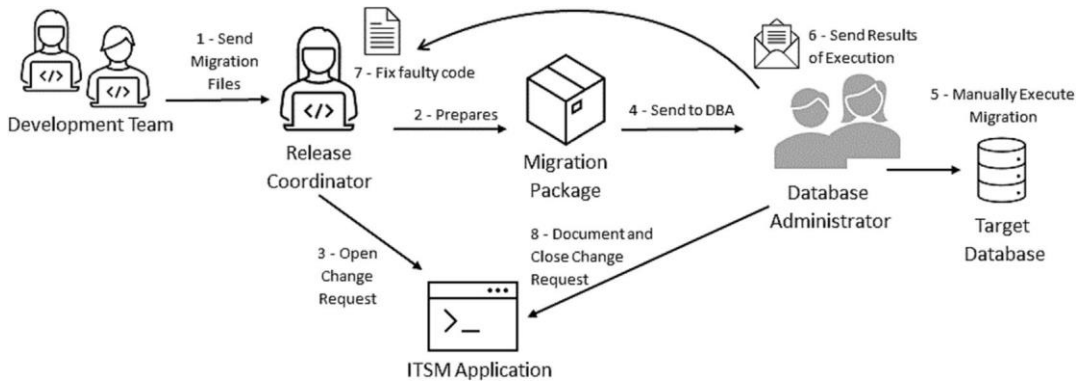

Рисунок 3.3 – Робочий процес розробки DWH до автоматизації

Робочий процес координував центральний координатор релізу, який збирає всі зміни від розробників і готує єдиний пакет розгортання, який вона надсилає електронною поштою адміністратору бази даних (DBA). Адміністратор бази даних виконує пакет міграції вручну для цільової бази даних. Команда розробників дотримується організації Scaled-Agile Framework, працюючи

спринтами по два тижні та щоквартально координуючи своє планування з усіма іншими командами розробників під час сеансу планування.

DWH не має автоматизованих тестів. Це призводить до того, що приблизно 25% потенціалу команди використовується для ручного тестування. Щоб переконатися, що конвеєри ETL у DWH працюють правильно, вони запускають повний тест раз на тиждень, який займає близько 8 годин. Під час цього тестового вікна розробники мають час, щоб перевірити вручну, чи їхні завдання створили правильний вихід даних. Однак немає механізму, який би підтвердив, що код все ще працює так само, як і до впровадження змін.

Сховище даних складається з рівня доступу з представленнями поверх ядра сховища даних, яке служить абстракцією для доступу до даних для вітрин даних. Таким чином команда розробників має можливість рефакторити таблиці бази даних і конвеєри ETL, не втручаючись у функціональність доступу до даних вітрини даних.

Під час інтерв'ю команда розробників сховища даних повідомила про три основні проблеми управління проектом.

1. Сценарії міграції бази даних, які координатор випуску отримав від розробників, не були перевірені належним чином і викликали помилки під час виконання. Ця низька якість призвела до додаткової роботи для координатора випуску, який часто не знав контексту сценаріїв, що дуже ускладнювало налагодження.

2. Створення одного пакета міграції з усіх змін призвело до створення великого пакета міграції. Ризик виконання такої кількості змін разом був високим. Якщо одна зі змін у пакеті містила помилку, усі інші зміни мали чекати виправлення перед просуванням до подальших середовищ.

3. Відсутність автоматизованих тестів призвела до того, що розгортання було можливим лише раз на тиждень через ручне тестування, яке передбачає запуск усіх конвеєрів ETL, щоб переконатися, що все працює.

Розробники хочуть розділити зміни в базі даних, попередньо зібрані в єдиний пакет міграції координатором випуску, на менші одиниці – по одній для

кожної зміни. Автоматизований конвеєр дозволить їм розгортати частіше та зменшить ризик використання одного великого пакета міграції, який зазнає невдачі, якщо лише одна частина буде неправильною. Мета полягає в тому, щоб створити більше, менших і менш ризикованих змін. Крім того, у цьому проекті велика кількість розгортань еволюції схеми бази даних, що містять DDL і мову обробки даних (DML), які містять невиконуваний код. Розробники сподіваються, що автоматизовані конвеєри зменшать кількість невдалих розгортань і забезпечать більш надійне розгортання змін у всіх середовищах завдяки покращенню якості коду завдяки автоматизованим конвеєрам.

3.6 Варіант використання внутрішньої бази даних (BE)

Варіант використання 2 включає проект розробки бек-енд бази даних у управлінні з сімома розробниками, які кожні пару місяців впроваджують зміни у виробництво. Замовник визначає, коли має відбутися виробництво. Розробники щотижня розгортають свої тестові та інтеграційні середовища. Серверна програма бази даних складається з приблизно 600 таблиць, які використовують близько 130 ГБ пам’яті, і містить понад 1000 пакетів із бізнес-логікою. Робочий процес розгортання показано на рисунку 3.4.

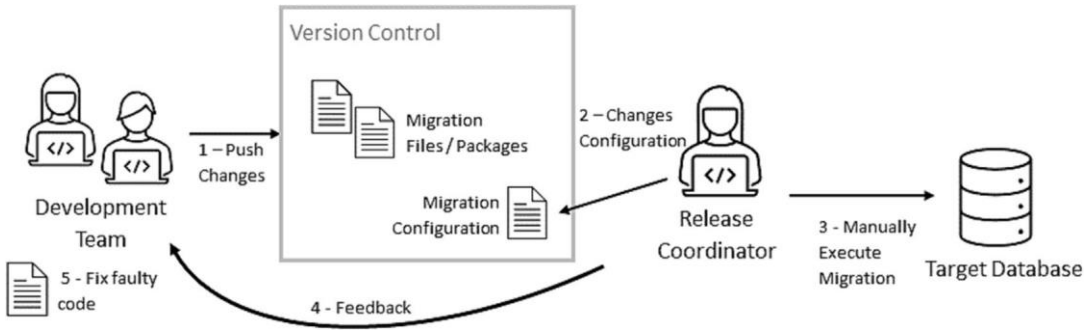

Рисунок 3.4 – Робочий процес розробки випадку використання 2 (BE) до автоматизації

Координатор випуску розгортає зміни зі своєї системи контролю версій і вручну виконує їх у цільовому середовищі. Оскільки робоча система доступна лише на сайті клієнта, доступ до якої здійснюється через VPN, середовище

інтеграції розробника служить для команди локальним робочим середовищем. Оскільки BE має автоматизовані тести, ручне тестування не потрібне. Однак один повний тестовий запуск вимагає до 3 годин, що не підходить для CI. Ось чому невеликий тестовий набір CI витягується з повного тестового набору для швидкого зворотного зв'язку. Повний набір тестів все ще виконується в неробочий час для регресійного тестування програми раз на день у вигляді нічного інтеграційного тесту.

Сервер бази даних містить рівень доступу для клієнтських програм. Цей рівень доступу генерується на основі механізмів відображення, де визначення подання відображаються на базові таблиці. Після кожної зміни бази даних цей рівень доступу створюється знову та також розгортається, щоб забезпечити функціональність програми-споживача.

Команда розробників організована за підходом Kanban, розставляючи завдання як окремі картки на дошці, завжди працюючи над картками з найвищим пріоритетом. Кожен член команди має працювати лише над однією карткою, доки її не буде реалізовано. Коли виникають нові завдання, учасники команди створюють нові картки та розставляють їх пріоритети під час коротких щоденних зустрічей.

Під час інтерв'ю команда бекенда бази даних повідомила про три типи проблем управління проектами.

1. Сценарії міграції бази даних, які координатор випуску отримав від розробників, тестувалися лише в контексті тестового середовища розробника й не включали зміни, внесені іншими розробниками. Це спричинило помилки під час виконання сценаріїв і ввело переробку сценаріїв змін.

2. Відсутність миттєвого відгуку про те, чи спрацювали сценарії змін, спричиняла багато перемикань контексту, коли координатору змін потрібна була інформація про невдалий сценарій.

3. Когнітивне навантаження на координатора релізу було високим, оскільки його робота містила завдання розробки та багато ручних завдань інтеграції, випуску та розгортання. Наявність кількох функцій у розробці

паралельно також збільшило когнітивне навантаження на розробників, оскільки вони завжди повинні були бути готові виправляти пакети, коли їх просувають.

Запровадивши автоматизований конвеєр, команда розробників сподівалася усунути потребу в локальній інтеграції. Коли зміни впроваджуються іншими членами команди, розробники повинні вручну інтегрувати ці зміни у своє локальне середовище розробки, що займає багато часу та є схильним до помилок. Оскільки вони не мають стандартного способу виконання інтеграції, кожне середовище розробки виглядає по-різному. Деякі розробники використовують контейнерні бази даних, інші – віртуальні машини. Крім того, розробники хочуть частіше інтегрувати зміни в центральне інтеграційне середовище. Зараз це робиться вручну координатором випуску раз на тиждень, що буває недостатньо часто та призводить до дрейфу версій бази даних у середовищах розробки.

3.7 Логістика для роздрібної торгівлі на базі даних (RET)

Варіант використання третій – це велика серверна система бази даних компанії роздрібної логістики. База даних діє як сховище даних і серверна частина бізнес-логіки для кількох програм, включаючи логістику, керування замовленнями та виставлення рахунків. Команда розробників складається з 35 розробників. Програма серверної частини бази даних складається з приблизно 5500 таблиць, 100 представлень і 300 пакетів з бізнес-логікою. Таблиці використовують близько 800 ГБ пам’яті.

Виробничу систему щовечора копіювали до тестового середовища, щоб мати набір реплік для розвантаження великих робочих навантажень. Однак він не слугував середовищем для створення нового коду до його розгортання у виробництві. На початку цього прикладу розробники входили безпосередньо в робочу систему, щоб змінити вихідний код системи. Вихідний код, що виконується у виробництві, потім щоночі зберігатиметься в системі контролю версій, щоб мати збережений знімок фактичного коду у виробництві. Однак

розробники не використовували цю кодову базу як еталон для своїх розгортань. Вони радше використовували живу систему як базову лінію.

Розгортання нещодавно розробленої функції або зміна коду відбулося одразу під час виробництва. Через високу навантаженість системи регулярно траплялося, що розгорталася процедура, яка наразі виконується, що призводило до блокування кешу бібліотеки. У цих випадках розробнику доводилося чекати, доки процедура завершить своє виконання, поки не можна буде розгорнути нову версію, що призводило до простою для розробника (див. рис. 3.5).

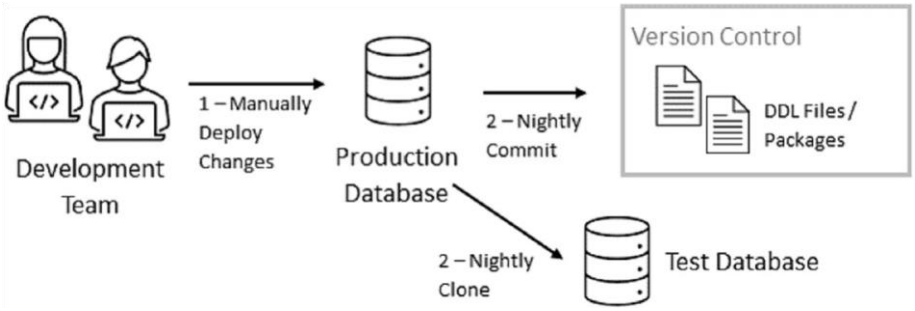

Рисунок 3.5 – Робочий процес розробки випадку використання 3 (RET) до автоматизації

Виробничий системний код щовечора передавався в контроль версій, щоб зберегти поточний стан виробництва. Однак розробники не використовували цей знімок як основу для змін коду. Через швидкі зміни вони покладалися на виробничий код як на базовий.

Серверна система бази даних не має рівня доступу, що призводить до прямого доступу до таблиці програмами-споживачами, що ускладнює міграцію бази даних.

Команда розробників працює гнучко, плануючи завдання під час щотижневих спринтів і координуючи поточну роботу під час щоденних зустрічей. Під час інтерв'ю команда розробників бази даних повідомила про наступні проблеми управління проектом.

1. При прямому виконанні сценаріїв у виробництві не було огляду змін або випусків. Лише переміщення даних узгоджувалися з центральною групою

адміністратора бази даних, але логічні зміни включали їх у виробництво безпосередньо, що ускладнювало пошук причин помилок, які виникали під час обслуговування, оскільки зміни не завжди повідомлялися.

2. Зміни в коді не перевірялися, доки вони не потрапили у виробництво, що призвело до помилок розгортання, які потрібно було негайно виправити.

3. Відстеження того, хто вніс які зміни в систему, було неможливим, оскільки весь виробничий код користувач системи передавав лише один раз на день у систему керування версіями.

Запроваджуючи систему контролю версій із конвеєром CI, команда розробників сподівалася підвищити видимість змін і запровадити підхід основного контролю версій для всього коду. Дотримуючись визначеного автоматизованого підходу до випуску коду, розробники хочуть підвищити впевненість під час розгортання змін у тому, що код є безпомилковим. За допомогою конвеєра CI вони також сподіваються збільшити автоматизований контроль якості, як автоматизоване тестування, статичний аналіз коду та генерація метрик коду. Розгортання змін у проміжному середовищі спочатку дає їм змогу запланувати розгортання на непікові години складу, зменшуючи ризик невдалого розгортання. Вони також хочуть запровадити процес схвалення для міграції даних, щоб команда DBA перевіряла їх перед запланованим.

3.8 Прийняття та оцінка конвеєра CI/CD

Незважаючи на відмінності між трьома сценаріями використання, описаними вище, тобто різними настройками команди, організацією, інфраструктурою та фінансовими ресурсами, інструментами та фреймворками, конвеєр CI/CD, запропонований нами, був прийнятгий у всіх з них.

Перед запровадженням конвеєра в усіх випадках використання необхідно було ввести такі кроки.

1. Налаштувати чітку структуру репозиторію контролю версій.

2. Визначити стратегію розгалуження та робочий процес розробки для сценаріїв міграції бази даних

3. Вибрати базу даних. інструмент міграції, який дозволяє автоматично застосовувати зміни до середовищ бази даних.

4. Встановлення вказівок щодо кодування та іменування та перевірка їх за допомогою інструменту статичного аналізу коду

5. Налаштування сервер CI, підключений до системи керування версіями, який запускає конвеєр і виконує інтеграцію та випуск.

У всіх випадках використання ми реалізували розроблений нами автоматизований CI та конвеєр розгортання (див. рис. 3.1), який запускається, коли відбуваються зміни в головній гілці репозиторіїв контролю версій. Таким чином команда швидко отримує відгуки та знає, чи їхні зміни працюють належним чином. Крім того, усі гілки функцій також інтегровані, надаючи безперервний відгук розробникам, чи їхній новий код придатний для виконання та відповідає стандартам кодування.

Усі інструменти, які використовуються для розробки та автоматизації, вже були присутні в компаніях як частина інфраструктури їх компанії, і їх не потрібно було оцінювати перед використанням. Щоразу, коли інструменти не інтегрувалися в конвеєр CI/CD за замовчуванням, для їх інтеграції писалися спеціальні сценарії. У всіх випадках використання.

Найскладнішою частиною реалізації конвеєра була концептуальна робота, тоді як кодування призвело до 500–2000 рядків коду сценаріїв. Результатом концептуальної роботи став набір інструментів і робочий процес розробки, який необхідно було спроектувати та оцінити.

3.9 Кількісний аналіз від впровадження CI/CD

Тепер ми кількісно оцінюємо переваги, отримані командами розробників завдяки введенню конвеєра. Було виміряно робочі процеси командного розгортання до та після впровадження. Крім того, інтерв’ю з командами

проводилися до та після впровадження конвеєра, щоб отримати додаткову інформацію про те, як автоматизація вплинула на команди. У таблиці 3.2 наведено вимірювання з трьох варіантів використання, зроблених до та після автоматизації.

Таблиця 3.2 – Порівняння трьох випадків використання до та після автоматизації

Вимірювання	DWH до	DWH після	BE до	BE після	RET до	RET після
Функції для кожного розгортання	5	1	1	1	1	1
Розгортання на тиждень на:						
- тест	1	20	8	13	n/a	n/a
- інтеграція	1	19	2	5	n/a	17
- виробництво	1	18	<= 1	<= 1	12	12
Люди, здатні розгорнути виробництво	1	30	1	1	35	35
% невдалих розгортань на:						
-тест	40%	9%	42%	9%	n/a	n/a
-інтеграція	32%	5%	15%	5%	n/a	8%
-виробництво	30%	3%	1%	1%	10%	2%
% код автоматично перевірено	0%	0%	62%	68%	0%	0%
Ручне тестування на тиждень у % від потужності команди	25%	25%	0%	0%	15%	15%
Час виконання	>= 7 днів	>= 7 днів	< 3 місяців	< 3 місяців	3 дні	4 дні
Час почати відновлення середовища	2 години	2 години	4 години	4 години	1 година	1 година

У варіанті використання DWH кількість функцій на розгортання зменшилася до однієї, оскільки розробники тепер можуть розгортати свої функції самостійно за допомогою конвеєра. У BE та RET розробники завжди застосовували розгортання однієї функції, тому кількість функцій на одне розгортання не змінюється.

Кількість щотижневих розгортань значно зросла у варіантах використання 1 і 2. Надання всім розробникам можливості самостійно виконувати розгортання

дало їм змогу активно просувати свої зміни по всьому конвеєру, що також допомогло збільшити кількість розгортань. Однак лише DWH дозволив усім розробникам виконувати робочі розгортання, тому кількість людей, які можуть виконувати робочі розгортання, зросла лише в цьому випадку використання.

BE міг виконувати робочі розгортання лише через VPN, наданий замовником і вимагав спеціальних прав доступу, тому лише провідний розробник міг виконувати робочі зміни. Випадок 3 має найбільші обмеження щодо середовища та розгортання, тому ми маємо лише вимірювання інтеграції після змін.

У всіх випадках розробники покладалися на конвеєри для інтеграції. Вони припинили вручну змінювати стан цільової бази даних, що також допомогло зменшити кількість невдалих розгортань, оскільки було усунено дрейф версій бази даних між середовищами. Це в основному пов’язано з більш частими автоматизованими інсталяціями в середовищі CI, де помилки виявляються рано. Нові конвеєри значно зменшили кількість невдалих розгортань у всіх трьох випадках використання.

Обсяг тестованого коду дещо збільшився для BE, але не збільшився для DWH і RET, оскільки DWH і RET не проводили автоматизоване тестування коду та не запроваджували його під час дослідження. Для DWH щотижневе ручне тестування споживає 25% потужності команди. Оскільки DWH не інвестував у створення автоматизованих тестів, цей % не покращився за час дослідження. RET також не інвестував у впровадження автоматизованого тестування, що призвело до 15% тестування вручну, що не покращилося з впровадженням конвеєра. Розробники UC2-BE, які використовують автоматичне тестування, не витрачають час на ручне тестування.

Оскільки DWH і BE використовували вікна фіксованого випуску, впровадження автоматизованих конвеєрів не вплинуло на час виконання. Цей вибір був зумовлений бізнес-рішенням обслуговувати клієнтів зі стабільною версією протягом певного часу перед внесенням змін. Цей фіксований час випуску впливає на щоденну роботу розробника та позбавляє можливості

використовувати невеликі цикли зворотного зв’язку та швидкі цикли тестування. У RET час виконання збільшився, імовірно, через те, що зміни більше не можна було вносити безпосередньо у виробництві, а їх потрібно було перевірити в системі контролю версій і пройти автоматизовану перевірку якості перед розгортанням. Очікується, що час виконання скоротиться, коли розробники звикнуть до нового процесу розгортання.

Автоматичне відновлення середовища після потенційного розгортання залишає його в пошкодженому стані. Через це час, необхідний для відновлення середовища, не покращився в жодному випадку використання. У всіх варіантах використання було вирішено використовувати гілки функцій замість чистої магістральної розробки, вимагаючи ручного об’єднання змін і дозволяючи членам команди розробляти досить великі набори змін паралельно до основної гілки. Комплексні зміни збільшують ризик невдалого розгортання або переробки, якщо зміни тих самих артефактів відбуваються на магістралі. Крім того, асинхронність цього робочого процесу негативно впливає на час виконання. Жодна з команд не зберігала набори змін у сховищі артефактів. Це порушує принцип найменших привілеїв для конвеєра розгортання, оскільки конвеєри CI/CD повинні мати лише мінімальний доступ, необхідний для виконання свого завдання. При повторному доступі до репозиторію керування джерелами цей принцип порушується.

3.10 Якісний аналіз запровадження автоматизації

Інтерв’ю, які ми провели з розробниками в DWH і BE перед запровадженням конвеєра, показали, що розгортання вручну передбачало високий рівень перемикання контексту, оскільки координатору випуску потрібно було зв’язатися з розробниками, якщо інсталяції не вдалися, через що вони призупиняли свої поточні роботи щодо виправлення помилок. Ця координація виправлення помилок призвела до додаткової роботи для координатора випуску та команди розробників. Збирання всіх змін в один

великий пакет міграції призвело до об’єднання їх в одну міграцію. Якщо одна зі змін була несправною, усі інші зміни повинні були чекати виправлення помилки, перш ніж їх просуватимуть через конвеєр. Це з’єднання призвело до розгортання у виробництво в неробочий час і, отже, до понаднормової роботи для команди. Вікна фіксованого випуску запобігали проходженню змін через конвеєр поза межами встановленого часу випуску, вимагаючи від розробників відстежувати розгортання своїх змін. Для всіх трьох варіантів використання відсутність стабільного середовища CI для інтеграції всіх змін і візуалізації впливу змін призвело до виявлення помилок на пізньому етапі конвеєра в тестовому середовищі та необхідності переробки, коли помилки виявлялися на пізньому етапі робочого процесу. Нарешті, в DWH і BE координатор випуску був єдиною точкою збою – якщо цієї людини не було поруч, зміни не потрапляли у виробництво. У RET розробники повідомили про високий рівень стресу під час безпосередньої зміни коду у виробництві. Їхньою метою було зменшити цей стрес, щоб зміни можна було впроваджувати з упевненістю. Через пряму зміну виробництва в RET розробники повідомили, що процес випуску неорганізований, оскільки вони не знали, що було змінено і коли.

Зворотний зв’язок із трьома сценаріями використання після впровадження конвеєра показав, що розробники та координатори випусків відчувають легше когнітивне навантаження в щоденній роботі. Тепер вони можуть зосередитися на розробці та довіряти конвеєру для правильного розгортання змін. Вони також зазначили, що їх повсякденна робота стала більш продуктивною, оскільки їх не заважали інтеграції та усунення несправностей у розгортанні так часто, як раніше, усунувши необхідність частого перемикання контексту. Усі випадки використання повідомили про покращення якості коду, оскільки помилки виявляються на ранній стадії під час розгортання CI конвеєра. Усі випадки використання повідомили, що за допомогою інструментів міграції бази даних, які надають Flyway і Liquibase, вони можуть інтегрувати перевірки після міграції в робочий процес розгортання, що виконується після кожної міграції.

Найбільшою проблемою, яку зараз бачать розробники в DWH і BE, є вікна фіксованого випуску, які не дозволяють їм просувати зміни у виробництво після завершення функцій. Це призводить до появи функцій, які очікують на випуск, створюючи мережу залежностей між нещодавно розробленими функціями. DWH і RET повідомили, що вони планують розпочати впровадження автоматизованого тестування, щоб більше не покладатися на ручне тестування, яке не забезпечує регресії. BE повідомили, що вони хотіли б скоротити проміжок часу між робочими випусками для більш частого розгортання у виробництві. Для RET новий процес розробки вважається викликом для розробників, оскільки використання системи контролю версій разом із процесом нового випуску вводить потребу навчання.

3.11 Вплив CI/CD

Питання дослідження стосувалися використання CI/CD для розробки додатків баз даних, включаючи проблеми та переваги. Найважливіші висновки полягають у тому, що практики CI/CD не мають широкого застосування в розробці додатків баз даних, за винятком великих програмних компаній, які продають системи баз даних, як-от SAP і MongoDB Inc та інші. Ми не знайшли повідомлень про конвеєри CI/CD, які використовуються в менших організаціях. Використання CI/CD не є широко поширеним і є складним через поєднання концептуальної, технічної та організаційної складності розробки додатків бази даних. Ці висновки змусили припустити, що запровадження CI/CD у промисловому контексті буде корисним. Щоб перевірити цю гіпотезу, було розроблено конвеєр CI/CD, представлений вище і опрацьовано відомості про його реалізацію та перевірку в трьох випадках використання. Нижче обговримо вплив, який мала реалізація практики CI/CD у трьох реальних проектах додатків баз даних, що відповідає задачі дослідження 3, сформульованій у вступі.

Наше дослідження показує, що автоматизовані конвеєри підвищують ефективність розробки бази даних і зменшують ризик встановлення змін у

виробництво. Вимірювання показали, що кількість розгортань збільшилася завдяки автоматизованому конвеєру CI, а кількість невдалих розгортань зменшилася, що робить розгортання більш надійними.

Крім того, інтеграція та автоматизація розгортання покращили роботу розробників у всіх випадках використання. Інтерв'ю показало, що розробники тепер мають менше паралельних завдань і можуть тестувати функції незалежно. Миттєвий зворотний зв'язок, який розробники отримують від CI, підвищує довіру до розгортання функцій у виробництві. Можливість розгортати функції для всіх розробників також видалила координатора випуску як єдину точку відмови та залежність для команд у двох випадках використання. У всіх трьох випадках використання впевненість розробників у додаванні змін зросла, оскільки вони могли покладатися на автоматизований конвеєр для їх ретельного тестування, що зменшувало когнітивне навантаження.

Однак у прикладі також показано кілька пунктів дій, які можуть ще більше покращити робочий процес розробки, а саме:

- видалення фіксованих дат випуску. Вимірювання показують, що час виконання між старим процесом розгортання та новим у DWH та BE не покращився. Це викликано організаційними рішеннями розгортати виробничі зміни у фіксовані дати випуску. Іншою причиною незмінного часу виконання можуть бути зміни, внесені вручну, які уповільнюють розгортання виробництва, що потребує часу розробників. Видалення вікон випуску релізів зменшить час виконання;
- автоматизація стратегії відновлення. Час відновлення середовища не змінився, оскільки це завдання не було автоматизовано в жодному випадку використання. Поточні налаштування CI/CD не включають механізм автоматичного відновлення або відновлення, який може бути запущений, якщо розгортання не вдається та залишає базу даних у пошкодженому стані. Таким чином, єдиний спосіб відновлення – повідомити адміністратора баз даних і дочекатися, поки механізм відновлення не запуститься вручну. Автоматизація механізму відновлення забезпечить додатковий рівень безпеки для команди.

Якщо помилки не можна виправити за допомогою прямої стратегії, це дозволить розробникам відновити середовище без залежності від команди DBA;

- магістральна розробка. Розгалуження функцій дозволяють командам і розробникам працювати незалежно, але є антипаттернами до CI та CD. Вони часто спричиняють конфлікти злиття, які важко вирішити та займають багато часу у рецензента запиту на злиття та у розробника. Антисинхронність гілок функцій призводить до довшого часу виконання. Гілки функцій також мають тенденцію до збільшення, що спричинено тривалим часом розробки. З робочим процесом git на основі магістралі розробники змушені будуть вносити більше незначних змін, які можна буде розгорнути окремо. Менші зміни призведуть до меншого розгортання та зниження ризику розгортання;
- покращення безпеки конвеєра розгортання. Можна використовувати репозиторій артефактів, де пакети розгортання можуть зберігатися після успішного випуску CI. Централізоване зберігання артефактів застосовувало б принцип найменших привілеїв до конвеєра розгортання. Конвеєр розгортання матиме доступ лише до попередньо протестованих версій, а не лише до останнього стану гілки функції.

Крім того, команди розробників повідомили про низку проблем під час реалізації конвеєра. Погодження стандартного робочого процесу git і узгодження інструкцій з кодування були одними з найбільших проблем. Запровадження тестування бази даних як частини розробки було ще одним викликом, оскільки воно змінило рутину розробників і потребувало вивчення та використання тестової структури. Організаційні проблеми включали потребу в дозволах для запуску робочих розгортань з автоматизованого конвеєра та необхідність довіряти всім розробникам просувати свої зміни через конвеєр без центрального координатора. Усі випадки використання повідомляли, що розробка та реалізація робочого процесу для об’єктів бази даних контролю версій були складними. У всіх трьох випадках використання сценаріїв використовувалося для інтеграції робочого процесу розробки з контролем версій для експорту DDL об’єктів бази даних і налаштування завдань розгортання.

Незважаючи на труднощі, з якими зіткнулися команди, у всіх випадках використання команди повідомили, що переваги конвеєрного впровадження перевищили їхні очікування та що автоматизація принесла додаткову цінність, як-от покращення якості коду, яку вони не очікували отримати.

Для розробки програмного забезпечення це означає, що в майбутньому слід збільшити інвестиції в автоматизацію змін бази даних, щоб отримати прибуток від позитивних ефектів конвеєрів CI/CD бази даних. У міру того, як автоматизація використовується все частіше, інструменти CI/CD бази даних повинні вдосконалюватися, а стандартизовані процеси та методи стануть доступними.

Була опрацьована інформація про три приклади промислового використання, усі з яких покладаються на систему реляційних баз даних Oracle, тому нашу роботу оцінювали лише в контексті додатків реляційних баз даних. Ми очікуємо, що розробка бази даних NoSQL буде подібною, але це потрібно дослідити на практиці. Було б цікаво дослідити, як інші технології баз даних, які часто не мають широкого інструментарію та покладаються на фреймворки з відкритим кодом, підтримують конвеєри CI/CD і з якими проблемами вони стикаються.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Методи та засоби психофізіологічного розвантаження як допоміжний процес в розробці ПЗ

Для галузі ІТ інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [34].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [36]. Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

1) надмірне інформаційне навантаження, суть якого полягає у тому, що кількість корисної інформації, яка надходить до мережі, перевищує психофізіологічні можливості її сприйняття людиною;

2) велика кількість інформації, яка сприймається, але не є корисною для фахівців в даний момент;

3) інформаційний голод, причиною якого є саме надлишок інформації, викликаний інформаційним перенавантаженням;

4) "інформоманія" як хвороба людини, яка робить останню знеособленою, залежною від перебування в інформаційному просторі і роботи з комп’ютером і чому вона віддає перевагу, уникаючи "живого" спілкування з людьми;

5) поява "кіберспільнот", що за своїми соціокультурними характеристиками набагато ближчі до представників інших культур у

глобальному інформаційному просторі, ніж до своєї етнонаціональної спільноти чи решти населення, не охопленого Інтернетом;

6) індивідуалізм і дегуманізація способу життя "мешканців" Інтернету – відсутність готовності ділитися своїми знаннями.

Слід розуміти, що комп'ютерні технології істотно впливають на життєдіяльність людини, припускаючи глобалізацію і технократизацію суспільства. Але в ще більшій мірі цей вплив поширюється безпосередньо на центральну нервову систему, яка звикає працювати в дуже інтенсивному режимі багатозадачності, де вже переважають не тривалі логічні роздуми, а інтуїтивно-реактивні ланцюжки розумових формулювань у зв'язку з величезним обсягом оброблюваної щодня інформації, кількість якої зростає за експоненціальною швидкістю. Виникає припущення, що саме збільшення обсягу інформації та прискорення її обробки людиною може згубно вплинути на розвиток розумових здібностей людини.

У [34] наведено перелік протипоказань з боку органів зору та загальних (соматичних) протипоказань, які забороняють роботу на ЕОМ, а також комплекс вправ для поліпшення здоров’я і підвищення працездатності.

Таким чином, за умови високого рівня робіт з ЕОМ рекомендується психофізіологічне розвантаження у спеціально обладнаних приміщеннях (кімнати психофізіологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов’язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ зі словесним самонавіюванням.

У рекомендованому сеансі, який має проводитися у кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим оформленням, виділяють такі три періоди, або фази:

Перший – абстрагування працівників від виробничої обстановки – відповідає фазі залишкового збудження. Звучить повільна мелодійна музика, пташиний спів. Обравши зручну позу, працівники адаптуються і психологічно готуються до наступних періодів.

Другий – заспокоєння – відповідає фазі відновлювального гальмування. Пропонується показ фото слайдів із зображеннями квітучого лугу, березового гаю, гладенької поверхні ставка тощо. Через навушники транслюється спокійна музика.

Як функціональне освітлення застосовують зелене світло. Яскравість світла має поступово знижуватися впродовж періоду заспокоєння, а наприкінці його світло вимикається зовсім на 1– 2 хв. Екран теж гасне.

Третій – активізація – відповідає фазі підвищеної збудженості.

На початку періоду світло вимкнене, через певний час на екрані з'являється червона пляма, розміри й яскравість якої поступово збільшуються.

Наприкінці періоду звучить бадьора музика. Вимовляються тричі мобілізуючі формули аутогенного тренування, яким мають передувати глибокий вдих та довгий глибокий видих.

Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являється бадьорість, хороший настрій. Загальний стан відчутно поліпшується.

4.2 Попередження аварій на виробництвах із застосуванням хлору

Хлор є частиною таблиці хімічних елементів і розташовується в ній під номером 17. У природі він зустрічається виключно у формі газу. Найчастіше він має специфічний зелений з жовтим переливом колір. Цей елемент важчий за повітря в 2,5 рази, тому накопичується в підвалах будинків, а на пересіченій місцевості в ярах і низинах. У воді ж хлор розчиняється без сліду і його наявність помітно тільки при великій концентрації (за рахунок специфічного запаху) [35].

В організмі людини в середньому міститься 95 г хлору. За добу людина споживає 5-10 г хлору (кухонна сіль). Він потрібен для вироблення в шлунку соляної кислоти, яка сприяє травленню і знищенню хвороботворних бактерій. Добова потреба хлору для людини становить 800 мг.

Хлор широко застосовується на виробництві, на його основі виготовляють отрутохімікати, розчинники, засоби для дезінфекції та миття, медикаменти. Хлор використовується в кольоровій металургії, у виготовленні пластмас тощо. Також хлор з успіхом застосовується і в побуті для очищення, відбілювання, прання. Завдяки незначним витратам і досить високої ефективності дезінфекції, хлор активно використовується для очищення і знезараження води в плавальних басейнах і питної водопровідної води.

Отруєння хлором можливе в разі:

– перевищення максимально допустимих концентрацій хлору для знезараження води в трубопроводі (сильний запах хлору);

– наявність хлору у великій кількості у воді басейну і часте купання в ньому;

– відбілювання і прання в закритому не провітрюваному приміщенні;

– аварії на підприємстві;

– використання хлору в якості зброї масового ураження.

В організм хлор потрапляє через слизові оболонки дихальної і травної систем, шкіру.

Ознаки отруєння хлором. До перших ознаках отруєння хлором відносяться:

– дискомфорт і подразнення слизової дихальних шляхів;

– підвищене слиновиділення і спазм голосових зв’язок;

– кашель і утруднене дихання;

– відчуття різі та печіння в очах, сльозотеча;

– нудота і гіркота у роті;

– головні болі і можливі судоми.

При попаданні на шкірний покрив або слизові спостерігається значний свербіж і гіперемія (почервоніння), вірогідні підшкірні крововиливи без пошкодження цілісності шкіри.

Тяжкість патологічного процесу та симптоми отруєння хлором знаходяться в прямій залежності від дози отруйної речовини (хлору) і тривалості його дії.

До прибуття медиків слід надати домедичну допомогу потерпілому:

– усунути джерело надходження отрути в організм – вивести або винести потерпілого поза зону дії отруйної речовини. При цьому необхідно пам’ятати про безпеку рятувальника – застосування марлевої маски або респіратора.

– забезпечити доступ чистого повітря;

– зняти забруднений одяг і теплою (не гарячою) водою промити контактуючі ділянки шкіри.

– у разі перорального надходження (проковтування) хлорвмісних рідин, потрібно промити шлунок. Промивати краще через зонд, або можна викликати блювання після рясного пиття.

– у разі пошкодження очей, промивання великою кількістю води або слабким розчином соди для зняття подразнення;

– полоскання ротової порожнини та носа содовими розчинами для мінімізації ушкодження слизових оболонок, застосування інгаляцій з додаванням соди для полегшення кашлю.

До профілактичних заходів отруєння хлором належать:

– забезпечення належних умов праці відповідно до санітарно-технічних вимог (вентиляція, провітрювання, справне обладнання);

– використання індивідуальних засобів захисту при роботі з хімікатами на виробництві;

– регулярні перевірки концентрацій хлору в повітрі робочої зони;

– проведення профілактичних медичних оглядів для виявлення схильності (доклінічних форм) і хронічних захворювань;

– дотримання вимог безпеки у використанні хлорвмісних рідин в побуті.

Суб'єкт господарської діяльності зобов'язаний забезпечити працівників хлорних об'єктів спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту відповідно до Положення про порядок забезпечення працівників спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту:

а) для захисту органів дихання – фільтруючими протигазами, ізолюючими дихальними апаратами та ізолюючими костюмами;

б) для захисту очей – захисними окулярами;

в) для захисту шкіри від їдких речовин – гумовими або прогумованими рукавицями, гумовими чоботами або шкіряними черевиками, сукняними костюмами.

Враховуючи значний ризик і широке застосування хлору, тяжкість ураження і високу можливість летального наслідку, у кожного повинен бути сформований алгоритм дій і чітка позиція – попередити отруєння легше і доцільніше, ніж лікувати і боротися з його наслідками.

ВИСНОВКИ

У цій роботі проаналізовано практики CI/CD для додатків баз даних. На основі огляду літератури ми повідомили про поточний стан впровадження, допоміжні програмні засоби, проблеми та передумови для впровадження практик CI/CD у розробці додатків баз даних. Запропоновано загальний конвеєр бази даних CI/CD, який об'єднує етапи забезпечення якості, інтеграції та розгортання бази даних. Досліджено три випадки реального використання розробки складної системи баз даних: один – агрегування даних у сховищі даних, другий – проект розробки серверної бази даних і один – серверна база даних у середовищі роздрібної торгівлі. Розроблений план конвеєра використовувався для автоматизації CI/CD у цих трьох випадках використання. У всіх випадках було спочатку вивчено методи розробки та розгортання командами розробників. Проаналізовано дані про опитування розробників, щоб дізнатися їхні погляди на поточну практику роботи та больові точки, а також щоби встановити важливі характеристики CI/CD. Потім ми запровадили запропонований конвеєр CI/CD, виміряли та опитали команди програмного забезпечення, щоб отримати відгук. Потім ми повідомили про переваги та проблеми автоматизації процесу інтеграції та доставки для програм баз даних.

З кількісної точки зору кількість невдалих розгортань було зменшено в усіх випадках використання. Стабільність систем підвищилася, що зробило операції більш простими. Також зросла кількість розгортань, частіше тестуючи виконання змін бази даних. Однак, якщо встановлені бізнесом фіксовані періоди випуску, це негативно впливає на робочий процес розробки, оскільки час виконання не може змінитися з фіксованими часовими рамками розгортання. З якісної точки зору розумове навантаження команд розробників було зменшено завдяки виключенню ручних завдань, необхідних перед автоматизацією конвеєра. Окрім переліку переваг автоматизації, ми також висвітлили типові проблеми, з якими стикаються розробники, впроваджуючи CI/CD у свої проекти.

Загалом CI та CD забезпечують швидший зворотний зв’язок і зменшують когнітивне навантаження розробників. Конвеєр CI/CD, який ми визначили, може служити еталонною архітектурою для тих, хто хоче прийняти CI/CD для додатків баз даних. Зокрема, наші звіти про основні вимоги до інфраструктури та завдання, які необхідно виконати перед початком впровадження CI/CD: визначення стратегії контролю версій і налаштування репозиторію контролю версій, вибір інструменту міграції бази даних для автоматизованого виконання зміни бази даних, автоматизація тестування, довіра до службових сценаріїв і використання інструментів моніторингу, які відстежують зміни, зроблені автоматизацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zampetti F, Vassallo C, Panichella S, Canfora G, Gall H, Penta MD. An empirical characterization of bad practices in continuous integration. Empir Softw Eng. 2020;25(2): 1095-1135.

2. Петришин Я. Роль CI/CD у підвищенні ефективності та надійності програмного забезпечення / Петришин Я., Марцинюк Я. // VII МСНТК, 25-26 квітня 2024 року. – Т. : ТНТУ, 2024. – С. 333–334.

3. Готович В. А. Застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення / В. А. Готович, А. В. Мачужак // XI Міжнародна науково-практична конференція молодих учених та студентів „Актуальні задачі сучасних технологій“, 7-8 грудня 2022 року. – Т. : ТНТУ, 2022. – С. 131–132. – (Комп’ютерно-інформаційні технології та системи зв’язку).

4. Дослідження ефективності процесів CI/CD в гнучких технологіях розробки програмного забезпечення / А. Вивюрка, Л. Мариненко, О. Нога, Б. Хоміцький, Т. Ланевич // Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“, 6-7 грудня 2023 року. – Т. : ФОП Паляниця В. А., 2023. – С. 402–403. – (Комп’ютерно-інформаційні технології та системи зв’язку).

5. Fluri J, Fornari F, Pustulka E. Measuring the benefits of CI/CD practices for database application development. In: IEEE/ACM International Conference on Software and System Processes, ICSSP 2023, Melbourne, Australia, May 14-15, 2023. IEEE; 2023: 46-57.

6. Herrmann K, Voigt H, Behrend A, Rausch J, Lehner W. Living in parallel realities. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM; 2017.

7. Herrmann K, Voigt H, Pedersen TB, Lehner W. Multi-schema-version data management: data independence in the twenty-first century. The VLDB J. 2018;27(4): 547-571.

8. Vitz M. Consumer-Driven Contracts – Testen von Schnittstellen innerhalb einer Microservices-Architektur. INNOQ. URL: https://www.innoq.com/en/ articles/2016/09/ consumer-driven-contracts/ (date of access: 14.11.2024).

9. Winters T, Manshreck T, Wright H. Software Engineering at Google: Lessons Learned From Programming Over Time: O'Reilly Media, Incorporated; 2020.

10. Haftmann F, Kossmann D, Lo E. Parallel execution of test runs for database application systems. In: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005 Böhm K, Jensen CS, Haas LM, Kersten ML, Larson Per-AAke, Ooi BC, eds. ACM; 2005:589-600.

11. Binnig C, Kossmann D, Lo E. Testing database applications. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006 Chaudhuri S, Hristidis V, Polyzotis N, eds. ACM; 2006:739-741.

12. Cleve A, Brogneaux A-F, Hainaut J-L. A conceptual approach to database applications evolution. In: Conceptual Modeling – er 2010. Springer Berlin Heidelberg Springer; 2010; Berlin, Heidelberg:132-145.

13. Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. Information and Software technology, 57, 21-31.

14. Afonso A, da Silva A, Conte T, Martins P, Cavalcanti J, Garcia A. LESSQL: dealing with database schema changes in continuous deployment. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER) IEEE; 2020:138-148.

15. Delplanque J, Etien A, Anquetil N, Ducasse S. Recommendations for evolving relational databases. In: Advanced Information Systems Engineering. Springer International Publishing Springer; 2020; Cham:498-514.

16. Meurice L, Nagy C, Cleve A. Detecting and preventing program inconsistencies under database schema evolution. In: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS) IEEE; 2016:262-273.

17. Delplanque J, Etien A, Anquetil N, Ducasse S. Recommendations for evolving relational databases. In: Advanced Information Systems Engineering. Springer International Publishing Springer; 2020; Cham:498-514.

18. Campbell L, Majors C. Database Reliability Engineering: Designing and Operating resilient database systems. 1st ed.: O'Reilly Media, Inc.; 2017.

19. Rehmann K-T, Seo C, Hwang D, Truong BT, Boehm A, Lee DH. Performance monitoring in SAP Hana's continuous integration process. SIGMETRICS Perform Eval Rev. 2016;43(4):43-52.

20. Ingo H, Daly D. Automated system performance testing at MONGODB. In: Proceedings of the Workshop on Testing Database Systems, DBTest '20.

21. The DataOps Manifesto - Read The 18 DataOps Principles. DataOps Manifesto - 18 DataOps Principles. URL: https://dataopsmanifesto.org/en/ (date of access: 12.10.2024).

22. Schuler R, Czajkowski K, D'Arcy M, Tangmunarunkit H, Kesselman C. Towards co-evolution of data-centric ecosystems. In: 32nd International Conference on Scientific and Statistical Database Management. ACM; 2020.

23. Haftmann F, Kossmann D, Lo E. A framework for efficient regression tests on database applications. The VLDB J. 2007;16:145-164.

24. Holt V, Ramage M, Kear K, Heap N. The usage of best practices and procedures in the database community. Inform Syst. 2015;49:163-181.

25. Gobert, M., Nagy, C., Rocha, H., Demeyer, S., & Cleve, A. (2023). Best practices of testing database manipulation code. Information systems, 111, 102105.

26. Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery – A systematic literature review. Information and Software Technology, 82, 55-79.

27. Nanda A, Tierney B, Helskyaho H, Widlake M, Nuijten A. Real World SQL and PL/SQL: Advice From the Experts: McGraw Hill LLC; 2016.

28. Fowler M. bliki: Branch By Abstraction. martinfowler.com. URL: https://martinfowler.com/bliki/BranchByAbstraction.html (date of access: 11.11.2024).

29. Database Concepts. Oracle Help Center. URL: https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/CDBs-and-PDBs.html (date of access: 30.09.2024).

30. How to Export Data using SQL Developer. fw_error_www. URL: https://www.oracle.com/database/technologies/appdev/sqldev/export-intro-1.html (date of access: 30.09.2024).

31. Duvall P, Matyas SM, Glover A. Continuous Integration: Improving Software Quality and Reducing Risk (the Addison-Wesley Signature Series): AddisonWesley Professional; 2007.

32. Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE access, 5, 3909-3943.

33. Martin RC. Agile Software Development: Principles, Patterns, and Practices: Prentice Hall PTR; 2003.

34. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин МОЗ України від 10.12.1998 № 7. // Офіційний сайт Верховної Ради України. – [Електронний ресурс]. – Режим доступу https://zakon.rada.gov.ua/rada/show/v0007282-98

35. Бідяк О. Профілактика отруєння хлором. // Офіційний сайт управління держпраці в Тернопільській області. – [Електронний ресурс]. – Режим доступу https://te.dsp.gov.ua/profilaktyka-otruyennya-hlorom/

36. Lypak, H., Kunanets, N., Veretennikova, N., Matsiuk, H., Kramar, T., & Duda, O. (2023, October). An Information System Project Using Augmented Reality for a Small Local History Museum. In 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT) (pp. 1-4). IEEE. DOI: 10.1109/CSIT61576.2023.10324194
https://ieeexplore.ieee.org/abstract/document/10324194

37. Kunanets, N., Dobrovolska, V., Filippova, N., Parviz, K., Lypak, H., Duda, O., ... & Dubrovina, L. (2020, September). Designing the Repository of Documentary Cultural Heritage. In Conference on Computer Science and Information

Technologies (pp. 1034-1044). Cham: Springer International Publishing. Doi: 10.1007/978-3-030-63270-0_70

38. Duda, O., Pasichnyk, V., Lypak, H., ...Matsiuk, O., Mudrokha, V. Formation of integrated repositories of social and communication data by consolidating the resources of museums, libraries and archives in smart cities projects. CEUR Workshop Proceedings, 2021, 2870, pp. 1420–1430. URL: http://ceur-ws.org/Vol-2870/paper104.pdf

39. Lypak, H., Kunanets, N., Pasichnyk, V., Veretennikova, N. Digitization Project for Historical and Cultural Heritage. 2020 IEEE 15th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT 2020 - Proceedings, 2020, 2, pp. 194–198, URL: https://ieeexplore.ieee.org/document/9321993

40. Липак Г., Липак Т., Кунанець Н. Проєктування інформаційної системи на основі машинного навчання для збереження та класифікації артефактів документальної спадщини. Вісник Хмельницького національного університету: технічні науки. Т. 334 № 4 (2024). С. 176-182. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/351

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль,11-12 грудня 2024 року

УДК 004.4; 005.3; 005.8

Руслан Матящук, Тарас Постолюк
Тернопільський національний технічний університет імені Івана Пулюя

ЗНАЧЕННЯ ВПРОВАДЖЕННЯ SPI ДЛЯ ПРОГРАМНИХ ПРОЄКТІВ

Ruslan Matiashchuk, Taras Postoliuk

THE IMPORTANCE OF SPI IMPLEMENTATION FOR SOFTWARE PROJECTS

Покращення процесів розробки програмного забезпечення (Software Process Improvement – SPI) має важливе значення для розробки програмного забезпечення, оскільки воно зосереджується на вдосконаленні та оптимізації процесів, пов'язаних зі створенням програмного забезпечення [1]. Основною метою SPI є підвищення якості, ефективності та надійності методів розробки програмного забезпечення, що зрештою призводить до кращих результатів як для розробників, так і для кінцевих користувачів.

SPI безпосередньо сприяє постачанню високоякісного програмного забезпечення з меншою кількістю дефектів. Коли процеси добре структуровані та ефективні, розробники можуть створювати більш надійне програмне забезпечення, яке відповідає або перевершує очікування клієнтів. Такий акцент на якості не тільки покращує репутацію команди розробників або компанії, але й зменшує довгострокові витрати, пов'язані з усуненням дефектів або проблемами продуктивності після розгортання.

Ефективність є ще одним важливим фактором SPI. Оптимізовані процеси забезпечують ефективне використання таких ресурсів, як час, бюджет і робоча сила. Таким чином досягаємо скорочення циклів розробки, дозволяючи командам постачати продукти швидше та економніше. На сучасному мінливому ринку програмних продуктів здатність швидко створювати програмне забезпечення без шкоди для якості забезпечує значну конкурентну перевагу.

Індустрія програмного забезпечення характеризується швидким розвитком технологій, мінливими вимогами клієнтів і динамічними вимогами ринку. Удосконалені процеси допомагають командам ефективніше реагувати на ці зміни, будь то пристосування до нових технологій, інтеграція відгуків клієнтів або масштабування для виконання більших проектів.

Стандартизуючи процеси, команди можуть досягти передбачуваних результатів, зменшуючи ризики, пов'язані з мінливістю практики розробки. Стандартизовані процеси також покращують співпрацю між членами команди, гарантуючи, що всі дотримуються однакових методологій і розуміють свою роль у досягненні цілей проекту.

Структури SPI, такі як CMMI (Capability Maturity Model Integration) або ISO/IEC 15504 (SPICE) чи поточним діючим ISO/IEC TS 33061:2021, забезпечують структуровані підходи до досягнення цих покращень [2].

Таким чином, SPI є життєво важливим аспектом розробки програмного забезпечення. Це покращує якість, покращує ефективність, підтримує адаптивність, забезпечує послідовність і сприяє як задоволенню клієнтів, так і моральному духу команди. Організації, які інвестують у SPI, отримують конкурентну перевагу та мають кращі позиції для довгострокового успіху.

Література

1. Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., & Zubrow, D. (1994). Benefits of CMM-based software process improvement: Initial results. Software Engineering Institute, CMU/SEI-94-TR-13.

2. Mathiassen, L., Nielsen, P. A., & Pries-Heje, J. (2002). Learning SPI in practice (pp. 3-21). Addison-Wesley, Boston.