Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____

(повна назва)

Кафедра _____Системотехніки_____

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

ПОЯСНЮВАЛЬНА ЗАПИСКА

Рівень вищої освіти _____другий (магістерський)_____

Дослідження застосування автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення

(тема)

Виконав:

студент __II__ курсу, групи ІТПм-22-2_

_____Нетребін Ю.М._____

(прізвище, ініціали)

Спеціальність _____

_____122 «Комп'ютерні науки»__

(код і повна назва спеціальності)

Тип програми ___освітньо-професійна_

(освітньо-професійна або освітньо-наукова)

Освітня програма _____

_Інформаційні технології проектування

(повна назва освітньої програми)

Керівник: _____доц. Чорна О.С._____

(посада, прізвище, ініціали)

Допускається до захисту

Зав.кафедри _____ _____Гребеннік І.В.____

(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Системотехніки

Рівень вищої освіти другий (магістерський)

Спеціальність 122 «Комп'ютерні науки»

(код і повна назва)

Тип програми освітньо-професійна

(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні технології проектування

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

"_____" _____20___р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Нетребіну Юрію Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження застосування автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення

затверджена наказом університету від " 20 " __11___ 2023 р. № 1373СТ.

2. Термін здачі студентом роботи до екзаменаційної комісії __10.01.2024_____

3. Вихідні дані до роботи Проведений аналіз існуючих інструментів. Напрямки для інтеграції автоматизованих тестових конвеєрів у процеси CI/CD, пропозиції з оптимізації.

4. Перелік питань, що потрібно опрацювати у роботі

1. Огляд та обґрунтування безперервної розробки програмного забезпечення

2. Постановка задачі на дослідження.

3. Методи вирішення задачі.

4. Проектування тестового конвеєра.

5. Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій _____

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1).

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН
09 жовтня 2023 р. - 08 січня 2024 р.

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Огляд та обґрунтування безперервної розробки програмного забезпечення	09.10.23-16.10.23	виконано
2.	Аналіз застосування досліджуваних технологій в існуючих системах.	17.10.23-24.10.23	виконано
3.	Постановка задачі на дослідження.	25.10.23-01.11.23	виконано
4	Опис методів рішення задачі.	02.11.23-09.11.23	виконано
5.	Методи вирішення задачі.	10.11.23-17.11.23	виконано
6.	Аналіз характеристик.	18.11.23-25.11.23	виконано
7.	Проектування тестового конвеєра.	26.11.23-08.12.23	виконано
8.	Оформлення висновків.	09.12.23-11.12.23	виконано
9.	Оформлення матеріалів кваліфікаційної роботи.	12.12.23-22.12.23	виконано
10.	Подання кваліфікаційної роботи керівникові та її попередній захист	23.12.23-29.12.23	виконано
11.	Подання кваліфікаційної роботи на рецензування	10.01.24	виконано

Дата видачі завдання ___ _____ 20___ р.
Студент _____
(підпис)
Керівник роботи _____ доц. Чорна О.С.

РЕФЕРАТ

Робота містить: *76* сторінки, *12* рисунка, *10* таблиць, *1* додаток, *41* джерел. Графічна частина дипломної роботи містить *21* плакатів.

БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, БЕЗПЕРЕРВНА ДОСТАВКА, АВТОМАТИЗОВАНІ ТЕСТОВІ КОНВЕЄРИ, JENKINS, ОПТИМІЗАЦІЯ ПРОЦЕСІВ РОЗРОБКИ, ІНТЕГРАЦІЯ АВТОМАТИЗАЦІЇ, МАЙБУТНІ ТРЕНДИ В CI/CD, ІНТЕГРАЦІЯ, КЕЙС-СТАДІ

У цій кваліфікаційній роботі проведено аналіз застосування автоматизованих тестових конвеєрів у процесах безперервної розробки програмного забезпечення, з акцентом на використанні Jenkins та інших інструментів CI/CD. Об'єктом дослідження є застосування автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення, з основним акцентом на використанні Jenkins та інших інструментів CI/CD.

Предметом дослідження є процеси безперервної інтеграції та доставки в галузі розробки програмного забезпечення. Це включає в себе методи, практики та інструменти, які використовуються для організації, автоматизації та оптимізації процесів розробки, тестування, інтеграції та впровадження програмних продуктів.

Метою дослідження є аналіз та дослідження методів оптимізації процесів CI/CD з використанням автоматизованих тестових конвеєрів. Основним завданням є вивчення існуючих інструментів для автоматизації тестування та їх впливу на процеси CI/CD. Методи дослідження включають аналіз літературних джерел, аналіз практичних випадків використання Jenkins та інших інструментів CI/CD, проведення експериментів та аналіз отриманих даних.

У роботі детально розглянуто основні аспекти інтеграції та автоматизації в процесах розробки програмного забезпечення, а також надано увагу ефективності Jenkins та його можливостям інтеграції з різними системами.

Галузь застосування результатів дослідження полягає у вдосконаленні процесів розробки програмного забезпечення, зокрема, у впровадженні автоматизованих тестових конвеєрів для підвищення ефективності CI/CD.

ABSTRACT

Thesis contains: *76* pages, *12* images, *10* tables, *1* applications, *41* sources. Graphic part of the thesis contains *21* posters.

CONTINUOUS INTEGRATION, CONTINUOUS DELIVERY, AUTOMATED TESTING PIPELINES, JENKINS, DEVELOPMENT PROCESS OPTIMIZATION, AUTOMATION INTEGRATION, FUTURE TRENDS IN CI/CD, INTEGRATION, CASE STUDY

This diploma thesis conducts an analysis of the application of automated testing pipelines in the processes of continuous software development, with a focus on the use of Jenkins and other CI/CD tools. The object of the research is the application of automated testing pipelines in continuous software development, with a primary focus on the use of Jenkins and other CI/CD tools.

The subject of the research encompasses the processes of continuous integration and delivery in the field of software development. This includes methods, practices, and tools used for organizing, automating, and optimizing the processes of development, testing, integration, and deployment of software products.

The research aims to analyze and study methods for optimizing CI/CD processes using automated testing pipelines. The primary task is to examine existing tools for test automation and their impact on CI/CD processes. Research methods include literature review, analysis of practical cases of using Jenkins and other CI/CD tools, conducting experiments, and analyzing obtained data.

In this work, we delve into the fundamental aspects of integration and automation within software development processes, with a specific focus on the efficiency of Jenkins and its integration capabilities with various systems. Additionally, we explore future trends in the field of CI/CD and gather user feedback on Jenkins.

The application of the research findings lies in enhancing software development processes, particularly in the implementation of automated testing pipelines to improve CI/CD efficiency.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення

Реліз – (англ. Release) готова версія програмного продукту, призначена для публічного використання.

CI – (англ. Continuous Integration) Безперервна інтеграція

CD – (англ. Continuous Delivery) Безперервна доставка

SLA – (англ. Service-level agreement) Угода про рівень послуг

DevSecOps – (англ. Development, Security, Operations) – Розробка, безпека, операції

Git – розподілена система керування версіями для відстеження змін у програмному коді та спільної роботи над ним.

Open-source – (з англ. Відкритий код) – Відноситься до програмного забезпечення, чий вихідний код доступний для перегляду, редагування та розповсюдження спільнотою.

Cloud-based – (з англ. Хмарний варіант) – Використовування інструментів або послуг, що надаються через хмарну інфраструктуру або віддалені сервери.

Self-hosted options – (з англ. Самостійні рішення) – Використовування інструментів або програмного забезпечення, які встановлюються та підтримуються на власних серверах користувача.

SaaS – (англ. Software as a service) – Програма як послуга

BDD – (англ. Behavior Driven Development) – Розробка на основі відтворення поведінки.

AI/ML – (англ. Artificial Intelligence/Machine Learning) – Штучний інтелект/Машинне навчання

Пайплайн – (англ. Pipeline) – Автоматизована послідовність обробки етапів або операцій у розробці програмного забезпечення.

CT – (англ. Continuous Testing) – Безперервне тестування.

DSL – (англ. Domain-specific language) – Предметно-орієнтована мова програмування

ВСТУП

Сучасна індустрія програмної інженерії вимагає надзвичайної швидкості та якості розробки програмного забезпечення. В умовах постійних змін та росту вимог користувачів, процеси розробки та доставки програмних продуктів стають важливішими, ніж будь-коли раніше. У цьому контексті безперервна інтеграція та доставка стають ключовими практиками для досягнення ефективності та високої якості розробки.

Актуальність роботи полягає у тому, що спостерігається зростаючий інтерес до методів та інструментів CI/CD серед організацій, які бажають покращити свої процеси розробки. Проте, вибір та реалізація ефективних практик CI/CD можуть бути складними завданнями. Тому дослідження у цій області має велику актуальність та потенціал для покращення якості розробки програмного забезпечення.

Метою даної дипломної роботи є аналіз та дослідження методів оптимізації процесів CI/CD з використанням автоматизованих тестових конвеєрів. Основним завданням є вивчення існуючих інструментів для автоматизації тестування та їх впливу на процеси CI/CD.

Об'єктом дослідження є процеси безперервної інтеграції та доставки у сфері розробки програмного забезпечення. Це охоплює методи, практики та інструменти, які застосовуються для організації, автоматизації та оптимізації процесів розробки, тестування, інтеграції та розгортання програмних продуктів.

Предметом дослідження є автоматизовані тестові конвеєри як інструмент оптимізації процесів CI/CD. В рамках предмету дослідження буде зосереджено увагу на методах та підходах до створення, впровадження та оптимізації автоматизованих тестових конвеєрів.

Для досягнення поставленої мети використовувалися наступні методи дослідження: аналіз наукової літератури, вивчення існуючих інструментів, практичні експерименти.

Одержані результати дослідження відзначаються новизною у вивченні та аналізі впливу автоматизованих тестових конвеєрів на процеси CI/CD у сфері розробки програмного забезпечення. Результати дослідження можуть бути використані для покращення процесів CI/CD в організаціях, що розробляють

програмне забезпечення. Аналіз інструментів та методів автоматизації тестування сприятиме підвищенню продуктивності та якості розробки.

Внаслідок проведеного дослідження було створено публікацію на тему "Огляд перспектив застосування автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення", де були розглянуті основні поняття безперервної розробки програмного забезпечення, вибір інструменту для автоматизації тестування, а також основні поняття автоматизації тестування [1].

1 ОГЛЯД ТА ОБҐРУНТУВАННЯ БЕЗПЕРЕРВНОЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Порівняння традиційної розробки з підходом CI/CD

Сучасний світ інформаційних технологій зазнає постійних змін та вдосконалень, і розробка програмного забезпечення (ПЗ) не є винятком. Однією з ключових складових сучасної розробки ПЗ є використання передових технологій та методологій для підвищення продуктивності, якості та швидкості розробки.

Проблема доставки програмного забезпечення полягає у тому, як найшвидше доставити користувачам ідею, якщо хтось її вже сформував.

Традиційні методології розробки програмного забезпечення, такі як модель Водоспад або модель V, часто передбачає ручні операції та підготовку до релізу, що вимагає багато часу та може призвести до помилок. Процеси ручного розгортання та конфігурації, а також відсутність автоматизації, роблять процес надто вразливим для помилок та важкодоступним для перевірки.

Основна проблема традиційного підходу полягає в тому, що цей процес не підтримує швидкість та часті релізи [1]. Існує велика втрата можливості у вигляді втрати прибутку, оскільки доходи з програми починають надходити лише після релізу. Отже, одна з основних мет– знайти способи скоротити час циклу, тобто час, який потрібний від моменту вирішення внести зміни (будь то виправлення помилки чи нова функція) до моменту, коли вони стають доступними для користувачів. Для досягнення цих мет поширюється підхід, що включає в себе часті та автоматизовані релізи програмного забезпечення. Чому саме автоматизація і частота релізів є ключовими?

Один з ключових принципів цього підходу– кожен внесок має потенціал стати релізом, і команда має бути готовою випускати його на будь-якому етапі. Автоматизація та часті релізи допомагають реалізувати цей принцип.

До вигоди цього підходу входять створення системи релізу, яка є систематичною, надійною та передбачуваною. Це призводить до значного скорочення часу циклу та швидкого впровадження функцій та виправлень

помилок до користувачів. Окрім цього, існують інші вигоди, які можуть бути передбаченими, а також приємними сюрпризами під час спостереження.

Серед них можливість команди автоматично розгортати будь-яку версію програмного забезпечення в будь-яке середовище. Тестери ПЗ можуть вибрати старі версії програми для перевірки змін в поведінці на нових версіях. Персонал технічної підтримки може розгортати випущену версію програми в середовищі для відтворення дефекту. Персонал оперативної підтримки може вибрати відомий справний збір, щоб розгорнути його на для відновлення під час аварійної зупинки. Релізи можуть виконуватися в один клік. Гнучкість, яку надають інструменти розгортання, змінює спосіб роботи команди на краще. Загалом, учасники команди більш контролюються над своєю роботою та збільшують ефективність.

Таблиця 1-1 – Порівняння традиційного підходу та CI/CD

Параметр	Традиційний підхід	CI/CD
Розробка	Кожна фаза завершується перед переходом до наступної	Постійна інтеграція та тестування протягом циклу розробки
Тестування	Окремий етап після завершення розробки	Автоматичні тести виконуються неперервно під час розробки
Цикл релізу	Програмне забезпечення розгортається тільки після завершення усіх робіт з розробки та тестування	Автоматизація та збірка дозволяє здійснювати часті та інкрементальні релізи
Автоматизація	Ручні процеси для інтеграції коду, тестування та розгортання	Автоматизація на протязі всього життєвого циклу розробки ПЗ
Співпраця	Відокремлення між командами розробки та операцій	Сприяє співпраці та спільній відповідальності між розробниками
Отримання відгуку від клієнтів	Обмежена участь та відгук від клієнтів	Постійне надання можливості отримувати частий відгук від клієнтів

1.1.1 Визначення DevOps

DevOps– це поєднання культурних філософій, практик та інструментів, яке збільшує здатність організації доставляти програми та послуги з великою швидкістю: еволюціонуючи та покращуючи продукти швидше, ніж організації, що використовують традиційні процеси розробки програмного забезпечення та управління інфраструктурою [2]. Ця швидкість дозволяє організаціям краще обслуговувати своїх клієнтів та більш ефективно конкурувати на ринку.

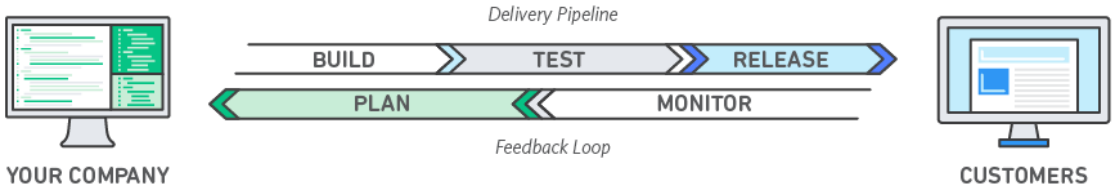

Рисунок 1-1- Модель DevOps

DevOps– це підхід до розробки програмного забезпечення, який об'єднує розробку (Development) та операції (Operations) для створення більш ефективних та автоматизованих процесів. Використання DevOps дозволяє організаціям розробляти та впроваджувати нові функції та оновлення швидше, зменшуючи ризики та збільшуючи якість продукту. Важливою частиною DevOps є автоматизація процесів розробки та впровадження, що допомагає зберігати високу швидкість доставки продукту на ринок.

Цей підхід допомагає підприємствам бути більш гнучкими, швидше реагувати на зміни на ринку та задовольняти потреби клієнтів шляхом постійного вдосконалення продуктів і послуг.

Важливою складовою методології DevOps є CI/CD, які допомагають реалізувати ключові її ідеї:

а) Посилення автоматизація– CI/CD сприяють автоматизації багатьох процесів розробки, включаючи тестування, збирання, розгортання та доставку програмного забезпечення. Ця автоматизація допомагає знизити ручні операції та помилки, що зазвичай виникають під час ручних процесів.

б) Швидкість та частота релізів– CI/CD дозволяють випускати зміни та оновлення програмного забезпечення швидше та регулярніше. Це відповідає ідеї DevOps щодо швидкості та невпинного вдосконалення.

в) Надійність– CI/CD сприяють підвищенню якості програмного забезпечення шляхом автоматизованого тестування та контролю якості. Це допомагає надійно доставляти програмне забезпечення при швидкому темпі розробки.

г) Масштабування та ефективність– CI/CD дозволяють керувати розробкою та розгортанням великих проектів та інфраструктур в ефективний та стандартизований спосіб. Це відповідає ідеї масштабу та ефективності в DevOps.

д) Співпраця– CI/CD допомагають об'єднати команди розробників та операціоністів в спільний процес. Вони спільно працюють над автоматизованими процесами, ділять відповідальності та стають більш взаємозалежними.

е) Безпека– CI/CD дозволяють впроваджувати безпеку в розробку, надаючи засоби для автоматичного тестування та перевірки безпеки в процесі CI/CD. Це відповідає концепції DevSecOps, де безпека стає неокремною частиною процесу розробки.

1.1.2 Визначення CI

CI– це практика розробки програмного забезпечення в рамках DevOps, де розробники регулярно об'єднують свої зміни коду в центральному репозиторії, після чого автоматизовані збірки та тести виконуються [3]. CI найчастіше стосується етапу збирання або інтеграції в процесі випуску програмного забезпечення і включає як складову автоматизації (наприклад, сервіс CI або збирання), так і культурну складову (навчання інтеграції на регулярній основі). Основні цілі безперервної інтеграції полягають у виявленні та виправленні помилок швидше, поліпшенні якості програмного забезпечення та зменшенні часу для перевірки та випуску нових оновлень.

Причиною для виникнення ідеї CI стало те що у минулому розробники команди могли працювати в ізоляції на протязі тривалого періоду часу і об'єднувати свої зміни в головну гілку лише після завершення роботи. Це робило процес об'єднання змін коду складним та затратним в часі, а також призводило до накопичення помилок на тривалий період без їхнього виправлення. Ці фактори робили процес швидкої доставки оновлень користувачам більш важким.

Розглянемо базовий процес CI. З безперервною інтеграцією розробники регулярно додають зміни в спільний репозиторій, використовуючи систему контролю версій, таку як Git. Перед кожною зміною розробники можуть запускати локальні модульні тести для перевірки коду як додаткового рівня підтвердження перед інтеграцією. Служба безперервної інтеграції автоматично збирає та виконує модульні тести для нових змін коду, щоб виявити будь-які помилки негайно.

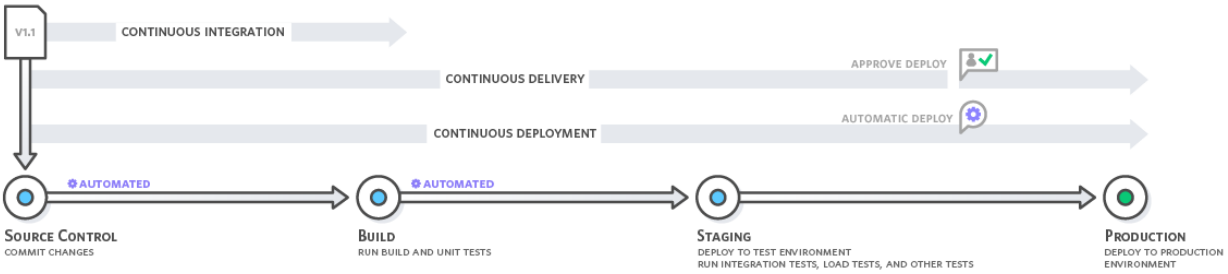

Рисунок 1-2 – Базова реалізація CI

Переваги безперервної інтеграції включають в себе:

– Збільшення продуктивності розробників шляхом автоматизації завдань і зменшення кількості помилок.

– Швидше виявлення та виправлення помилок завдяки частому тестуванню.

– Швидку доставку оновлень користувачам завдяки регулярним інтеграціям.

1.1.3 Визначення Continuous Delivery

CD– це практика розробки програмного забезпечення, де зміни в коді автоматично готуються до кінцевого випуску [4]. Ця практика розширює можливості безперервної інтеграції, розгортаючи всі зміни коду в тестове середовище і/або середовище використання після етапу збирання. Правильно реалізована безперервна доставка гарантує, що розробники завжди матимуть готовий до випуску збудований артефакт, який пройшов стандартизований процес тестування.

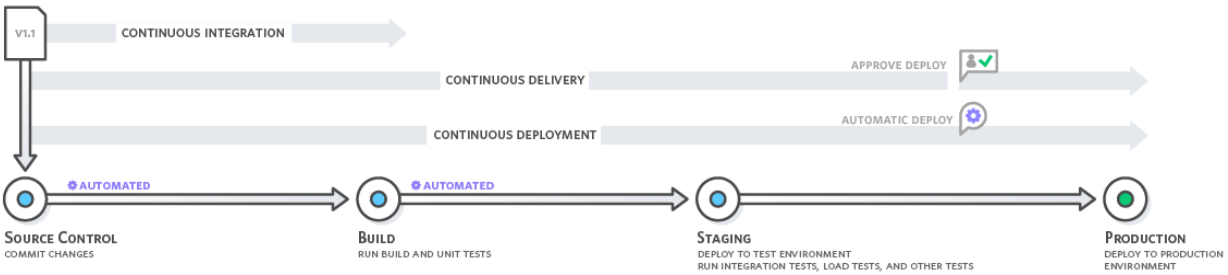

Рисунок 1-3- Базова реалізація CD

З Continuous Delivery кожну зміну коду збирають, тестують, а потім переносять на тестове середовище чи проміжне середовище перед випуском у робоче середовище. Безперервне розгортання (Continuous Deployment) відрізняється від безперервної доставки тим, що випуск у робоче середовище відбувається автоматично без вимагання схвалення вручну.

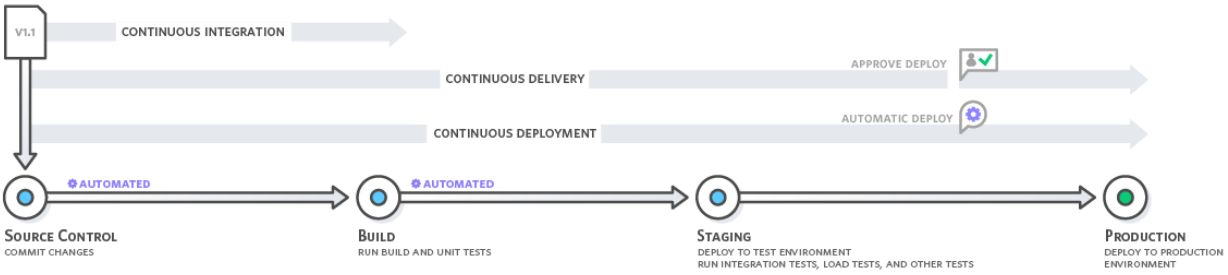

Рисунок 1-4- Безперервна доставка та безперервне розгортання

Таким чином безперервна доставка автоматизує весь процес релізу програмного забезпечення. Кожна зміна, яка фіксується, запускає автоматизований потік, який здійснює збірку, тестування, та підготовку

оновлення. Кінцеве рішення про розгортання до робочого середовища приймається розробником.

Переваги безперервної доставки:

– Автоматизація процесу випуску програмного забезпечення робить цей процес більш ефективним та швидким.

– Підвищення продуктивності розробників допомагають їм бути більш продуктивними, звільняючи їх від ручних завдань і спонукаючи до вчинків, які допомагають зменшити кількість помилок, які потрапляють до користувачів.

– Виявлення та виправлення помилок раніше дозволяє на ранніх стадіях розробки завдяки більш частому та більш глибокому тестуванню. Безперервна доставка дозволяє виконувати різноманітні типи тестів, оскільки весь процес є автоматизованим.

– Швидка доставка оновлень допомагає швидко та регулярно доставляти оновлення користувачам. Коли безперервна доставка реалізована належним чином, у розробників завжди буде готовий до випуску артефакт, який пройшов стандартизований процес тестування.

1.1.4 Типи інструментів CI/CD

Існують різні типи інструментів CI/CD, і їх вибір важливий для успішної реалізації процесів автоматизації [5]. Основні типи інструментів включають наступні фактори:

а) Open-source та комерційні рішення

Те, чи є інструмент CI/CD відкритим джерелом або комерційним рішенням, є важливим моментом. Відкрите програмне забезпечення безкоштовне у використанні, але підтримка зазвичай залежить від спільноти. Якщо ви виявите помилку або хотіли би додати новий функціонал, вам доведеться чекати, поки хтось інший вирішить це питання, або ж ви можете самостійно вносити зміни. Незалежно від того, яким чином вноситься новий код, це може зайняти певний час для перевірки та прийняття його керівниками проекту.

Навпаки, комерційні платформи зазвичай включають щодо технічної підтримки і надають регулярні оновлення з новими функціями та виправленнями помилок. Багато комерційних інструментів CI надають

безкоштовний рівень, але зазвичай існує обмеження на кількість одночасних збірок або кількість хвилин для збірки.

б) Cloud-based та self-hosted options

Хмарні або SaaS інструменти CI/CD зазвичай включають веб-інтерфейс для управління конвеєрами збірок, агенти або «ранери» для збірки розміщені на публічних або приватних хмарових інфраструктурах.

З хмарним рішенням немає потреби встановлювати чи підтримувати їх. Після реєстрації у вас зразу є доступ до CI-сервера і можливість запускати збірки на агентах, розміщених у хмарі без затримки. Деякі SaaS-варіанти також дозволяють виконувати збірки на ваших власних агентах за менші кошти.

Самостійні рішення дозволяють вибирати, де розміщені як ваш сервер збірки, так і агенти: в приватній хмарі, на обладнанні на місці або в публічному хмаровому середовищі. Компроміс полягає в тому, що ви відповідальні за управління встановленням та оновленнями CI-сервера та агентів. При розгортанні на місці вам також слід розглядати питання масштабування та доступності сервера та агентів.

в) Інтеграції

Конвеєр CI/CD торкається кожного етапу процесу розробки програмного забезпечення. Таким чином, CI/CD повинна мати можливість інтегруватися з кожним елементом інструментарію, включаючи систему контролю версій, систему відстеження задач, платформи для комунікації в команді та середовища розгортання, які розміщені в хмарі чи на місці.

Наявність інтеграцій варіює від інструменту CI. Деякі системи CI входять до існуючої екосистеми або обмежені певними системами управління джерелами або сервісами хостингу систем контролю версій. Інші інструменти надають можливість інтеграції з інструментами від різних постачальників і дають можливість розширити платформу за допомогою плагінів або API.

г) Конфігурація

Налаштування вашого автоматизованого конвеєра CI/CD включає в себе все, від визначення тригера, який ініціює кожний запуск конвеєра, до визначення поведінки у випадку невдалих збірок або тестів. Ці налаштування можна конфігурувати за допомогою сценаріїв або з використанням інтерфейсу користувача.

Використання інтерфейсу може бути простіше для тих, хто ще не знайомий із CI/CD та для непрограмістів, проте сценарії дозволяють

використовувати підхід конфігурації як коду, де логіка зберігається у системі контролю версій. Деякі інструменти також надають можливість генерації сценаріїв з інтерфейсу, даруючи вам можливість використовувати кращі практики обох підходів.

Таблиця 1-2 – Рейтинг CI

CI Система	Всього	Компанія	Персонально
GitHub Actions	53%	27%	42%
Jenkins	52%	47%	12%
Gitlab CI	35%	27%	17%
Azure DevOps	15%	13%	5%
CircleCI	13%	8%	6%
Travis CI	10%	3%	7%
Bitbucket Pipelines	10%	8%	4%
AWS CodePipeline / AWS CodeStar	10%	8%	3%
Custom tool	9%	7%	3%
TeamCity	8%	6%	2%
Google Cloud Build	6%	3%	3%
Bamboo	4%	4%	1%
JetBrains Space	4%	1%	3%
Drone	3%	2%	2%
AppVeyor	2%	1%	1%
GOCD	2%	1%	1%
Codeship	1%	1%	1%

Аналізуючи дані з таблиці, видно, що GitHub Actions є найбільш популярним інструментом для CI, використовуваним користувачами як в особистих, так і в корпоративних цілях. За результатами опитування [6,7], 53% усіх відповідей підтвердили їхнє регулярне використання GitHub Actions.

Загальною тенденцією є зростання популярності хмарних рішень для CI/CD. Понад половина респондентів (51%) вже перейшла на використання

хмарних інструментів для CI/CD. Ця тенденція свідчить про зростаючий інтерес до хмарних рішень, які можуть спрощувати процеси CI/CD і забезпечувати гнучкість.

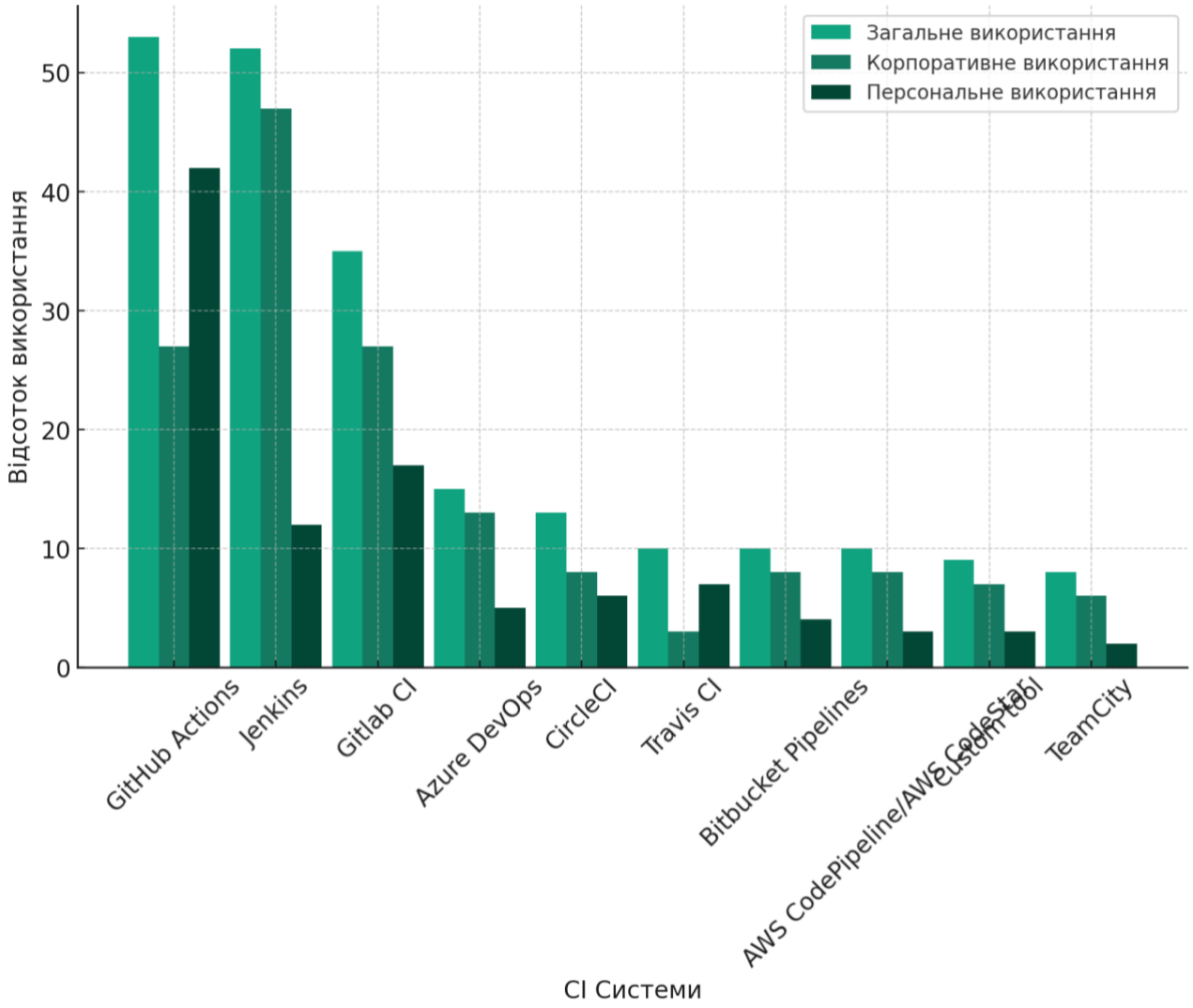

Рисунок 1-5 – Рейтинг використання CI/CD

У цілому, результати таблиці підтверджують значення CI/CD для розробників і показують, що вони активно використовують різноманітні інструменти для автоматизації розробки, тестування та релізу програмного забезпечення. Зростання популярності хмарних рішень також свідчить про поступову трансформацію процесу розробки та швидкість доставки програмного забезпечення.

1.2 Огляд інструментів для автоматизації тестування

Перш ніж робити огляд інструментів для автоматизації дамо наступні визначення:

Тестування програмного забезпечення– це процес аналізу програмного продукту для виявлення різниці між існуючими та потрібними умовами (тобто виявлення помилок), а також для перевірки функціональності продукту [8,9]. Тестування є важливим елементом загального процесу розробки програмного забезпечення і включає в себе запуск програми або системи з метою знайти помилки програмного забезпечення та переконатися, що продукт відповідає зазначеним вимогам.

Автоматизоване тестування– це використання спеціального програмного обладнання для контролю виконання тестів і порівняння фактичних результатів із передбачуваними. Автоматизація тестування може автоматично перевіряти, чи реальні результати відповідають очікуваним результатам, тим самим забезпечуючи ефективність тестування та допомагаючи у виявленні помилок.

Одним із стандартів, який регулює процеси тестування програмного забезпечення, є ISO/IEC/IEEE 29119. Цей набір стандартів включає положення та рекомендації щодо планування та управління тестуванням, документування процесів тестування, визначення тестових проектів, відбір тестових методів і багато іншого. Стандарт ISO/IEC/IEEE 29119 спрямований на підтримку міжнародних найкращих практик у тестуванні програмного забезпечення.

1.2.1 Популярні інструменти для автоматизації

Вибір інструменту для автоматизації тестування може бути складним, але орієнтування на популярність інструментів у спільноті розробників може бути корисним. Опитування проведене компанією Jetbrains [7] відображає переваги розробників та тестувальників щодо інструментів автоматизації тестування. Результати цього опитування показують, що такі інструменти як Postman, JUnit та Jest мають велику частку використання, що вказує на їх велику популярність і широкі можливості застосування.

Таблиця 1-3 – Рейтинг використання інструментів для тестування

Інструмент	Використання у 2022 р (%)	Попередній період (%)
Postman	34	29
JUnit	31	34
Jest	25	23
Pytest	20	15
Selenium WebDriver	19	20
Apache JMeter	15	11
Cypress	13	12
NUnit / xUnit.Net	11	9
Cucumber	10	8
Mocha	8	10
MSTest / VSTest	7	4
Custom	6	5
Playwright	5	1
SoapUI	5	4
TestNG	5	4
Allure	4	2
Robot Framework	4	2
Gatling	3	2
Behave	2	1
RSpec	2	4
ExtentReports	1	0
Katalon Studio	1	1
TestCafe	1	1
Інші	8	9
Не використовують	10	11

Ці дані демонструють динаміку ринку та переваги розробників. Можна звернути увагу на зростання популярності інструментів, таких як Pytest та Cypress, що може вказувати на збільшення уваги до автоматизації тестування JavaScript та Python відповідно, а також на високу динаміку змін у галузі.

Також треба звернути увагу на результати того ж опитування на такі популярні технології як Behavior Driven Development (BDD) та AI/ML.

BDD який є практикою розробки програмного забезпечення, що спрямована на поліпшення співпраці між розробниками, QA фахівцями та не-технічними учасниками проекту, такими як бізнес-аналітики. BDD зосереджується на створенні чітких і зрозумілих для всіх учасників проекту сценаріїв поведінки системи через їх опис в природній мові.

Згідно з даними опитування Jetbrains, більшість респондентів (83%) не використовують BDD в своїх проектах, що може вказувати на недостатню обізнаність або складнощі інтеграції BDD практик в існуючі робочі процеси. Проте, серед тих, хто використовує BDD, Cucumber є лідером. Цей інструмент надає платформу для виконання тестових сценаріїв, описаних мовою Gherkin, що дозволяє писати поведінкові тести на бізнес-орієнтованій мові.

Що до застосування штучного інтелекту та машинного навчання (ML) у сфері тестування відкриває нові перспективи для оптимізації та автоматизації робочих процесів. Згідно з опитуванням, 14% респондентів вже використовують AI/ML технології у своєму тестуванні у 2022 році, що на 6% більше, ніж у 2021 році. Це може включати застосування AI для виявлення шаблонів у даних тестування, прогнозування проблемних областей або автоматизації складних сценаріїв тестів.

1.2.2 Критерії вибору інструментів для автоматизації тестування

Автоматизація тестування є критично важливою для підтримки швидкого циклу розробки та забезпечення високої якості продукту [10]. Правильний вибір інструментів, який враховує специфіку проекту та команди, може значно підвищити ефективність розробки та забезпечити успішне впровадження продукту на ринок.

а) Сумісність з мовами програмування та технологіями

Сумісність інструментів тестування з мовами програмування та набором технологій проекту є одним із найважливіших факторів при виборі [11]. Інструмент має забезпечувати не тільки технічну сумісність, а й відповідність рівню тестів, які необхідно автоматизувати. Огляд популярних мов програмування та відповідних їм інструментів автоматизованого тестування для різних типів тестів наведено у наступній таблиці:

Таблиця 1-4- Інструменти в залежності від мови та рівня тестування

Мова програмування	Модульні тести	Інтеграційні тести	Системні тести	E2E тести
Java	JUnit, TestNG	JUnit, TestNG	TestNG, Selenium	Selenium

C#	NUnit, MSTest	NUnit, SpecFlow	MSTest, Selenium	Selenium, SpecFlow
Python	PyTest, unittest	PyTest	Robot Framework	Robot Framework
JavaScript	Jest, Mocha	Jest, Mocha	WebDriverIO	Cypress, WebDriverIO
Ruby	RSpec	Cucumber	Capybara	Cucumber, Capybara
PHP	PHPUnit	PHPUnit	Codeception	Codeception

Кожен з інструментів має свої унікальні особливості та переваги, які слід враховувати під час вибору:

– JUnit та TestNG вважаються стандартом для модульного тестування в Java-проектах, з великими можливостями для розширення та налаштування.

– NUnit та MSTest добре інтегровані в екосистему .NET, пропонуючи зручні інструменти для модульного тестування у проектах на C#.

– PyTest - це швидкий та гнучкий інструмент для тестування, який відмінно підходить для роботи з Python, особливо з його розширюваністю через плагіни.

– Jest та Mocha надають широкий спектр можливостей для тестування JavaScript-коду, від модульного до інтеграційного.

– Cypress і WebDriverIO пропонують сучасні підходи до E2E тестування веб-додатків, що дозволяє імітувати дії реальних користувачів.

– PHPUnit є де-факто стандартним вибором для тестування PHP-коду, тоді як Codeception покриває більш широкий спектр тестування, включно з приймальним та функціональним.

б) Можливість інтеграції з системами контролю версій та іншими інструментами CI/CD

Інтеграція інструментів автоматизації тестування з системами контролю версій та інструментами неперервної інтеграції та доставки (CI/CD) є критично важливим аспектом сучасної розробки програмного забезпечення. Ця синергія підвищує ефективність розробки, забезпечуючи швидке виявлення помилок, автоматизацію рутинних завдань і впровадження кращих практик якості коду. Детальніше розглянемо ключові аспекти цієї інтеграції:

– Центральним елементом будь-якого процесу розробки є система версій контролю версій, такі як Git. Інструменти автоматизації тестування мають підтримувати глибоку інтеграцію з такими системами, щоб забезпечити запуск тестів при здійсненні змінах у кодовій базі.

– Інструменти CI/CD дозволяє автоматизувати запуск тестових наборів, управління результатами тестування та звітність, а також гарантувати, що тести виконуються в контрольованих, стабільних середовищах.

– Сучасні тести часто виконуються в контейнерах (наприклад, Docker [12]) або віртуалізованих середовищах, що дозволяє забезпечити консистентність тестових умов. Інструменти тестування мають підтримувати ці технології для легкої інтеграції з процесами CI/CD.

– Інструменти тестування повинні дозволяти створювати шаблони для тестових пайплайнів, що спрощує інтеграцію та повторне використання тестових процесів у різних проектах та середовищах.

– Важливо, щоб інструменти мали гнучкі налаштування та API для розширення, щоб можна було адаптувати процеси тестування до специфіки проекту, інтегрувати користувацькі скрипти, плагіни, хуки тощо.

– Автоматизовані інструменти тестування мають включати можливості для детальної звітності про результати тестів та надсилання сповіщень за допомогою електронної пошти, або інших засобів комунікації, що забезпечує своєчасне виявлення та виправлення помилок.

– Необхідно забезпечити, що інтегровані інструменти відповідають вимогам безпеки та стандартам індустрії, таким як OWASP [13], ISO 27001, тощо, особливо у випадках роботи з конфіденційними даними.

в) Підтримка різних типів та рівнів тестування

Ефективна стратегія автоматизованого тестування вимагає від інструментів здатності охоплювати широкий спектр тестових процедур, від функціональних перевірок до оцінки нефункціональних аспектів. Це означає підтримку як різних типів тестування, так і різних рівнів перевірки програмного забезпечення.

Види тестування включають, але не обмежуються, наступними [14,15]:

– Функціональне – валідація відповідності функцій програми визначеним вимогам.

– Нефункціональне – перевірка продуктивності, надійності, безпеки та інших ключових характеристик системи.

– Регресійне – гарантування, що зміни в коді не порушили існуючу функціональність.

– Навантажувальне – оцінка поведінки системи під високим навантаженням.

– Тестування безпеки – виявлення потенційних вразливостей у програмі.

– UI тестування – перевірка елементів користувацького інтерфейсу на відповідність дизайну та інтуїтивність використання.

Рівні тестування забезпечують глибину аналізу та переконуються, що кожен аспект системи ретельно перевірений [16]:

– Модульне тестування – фокусується на окремих частинах програми, зазвичай на рівні класів або методів.

– Інтеграційне тестування – оцінює взаємодію між різними модулями або внутрішніми системами.

– Системне тестування – розглядає поведінку програми в цілому в умовах, які максимально наближені до реального використання.

– Тестування приймання – перевіряє, чи програма готова до випуску та відповідає бізнес-вимогам.

Інструменти автоматизації тестування повинні підтримувати інтеграцію цих типів та рівнів тестування у єдиний потік роботи, дозволяючи командам швидко переключатися між різними видами тестів та гарантувати, що кожен випуск програмного продукту відповідає високим стандартам якості.

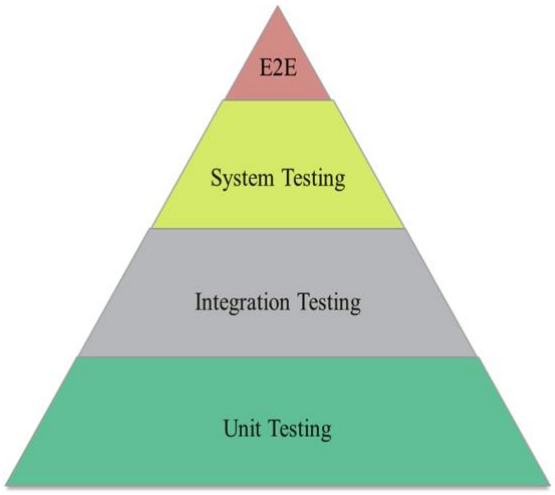

Рисунок 1-6- Піраміда рівнів тестування

г) Наявність готових рішень (плагінів, розширень)

Готові рішення, такі як плагіни та розширення, істотно збільшують функціональність інструментів автоматизації тестування та дозволяють налаштувати процес тестування під специфічні потреби проекту. Вони можуть полегшити інтеграцію з іншими інструментами, додавати нові типи тестів, покращувати звітність, автоматизувати аналіз коду, і навіть забезпечувати підтримку нових тестових фреймворків, які не були доступні в оригінальному інструменті.

Плагіни та розширення вносять велику гнучкість у процес тестування:

– Інтеграція з іншими інструментами дозволяє інструменту автоматизації тестування працювати з системами, які вже використовуються в організації, такими як системи управління базами даних, системи контролю версій, інструменти моніторингу і документування.

– Наявність спеціалізованих функцій дозволяє розширити стандартні можливості інструменту, щоб підтримати унікальні аспекти проекту, такі як сумісність зі специфічними пристроями або платформами.

– Аналіз коду та звітність можуть забезпечити додаткові можливості для аналізу якості коду, генерації звітів про покриття коду тестами, статистику про помилки та тренди в роботі команди.

Ось декілька прикладів плагінів та розширень для популярних інструментів автоматизації тестування:

Таблиця 1-5 – Розширення до інструментів тестування

Інструмент	Плагін або розширення	Опис функціоналу
Selenium	WebDriver	Уможливлює автоматизацію дій в веб-браузері.
Jenkins	Cobertura	Візуалізує покриття коду тестами.
JUnit	Mockito	Підтримка створення мок-об'єктів для модульного тестування.
TestNG	ReportNG	Надає звіти у зручному HTML форматі.
PyTest	PyTest-Django	Дозволяє використовувати PyTest для тестування Django проектів.
Mocha	Chai	Додає більше можливостей для асерцій у тестах.

Використання плагінів та розширень може значно спростити входження нових членів команди в проект, оптимізувати вже існуючі процеси, а також підвищити якість і швидкість розробки продукту. При виборі інструментів для автоматизації важливо враховувати не тільки їх основні можливості, а й доступність розширень, які можуть бути необхідні для задоволення унікальних потреб вашого проекту.

д) Легкість використання та обслуговування

Інструмент автоматизації тестування повинен забезпечувати легкість використання, що є ключовим фактором для ефективності та продуктивності розробників і QA-інженерів. Інтуїтивний інтерфейс зробить інструмент більш доступним для новачків, дозволяючи їм швидше включитися в робочий процес і зменшуючи потребу в тривалому навчанні. Наприклад, інструменти з графічними інтерфейсами користувача, які надають візуальне створення тестів, можуть спростити розробку тестових сценаріїв без глибокого знання коду.

Інструменти повинні також пропонувати просте управління залежностями та конфігурацією, щоб мінімізувати технічні перепони. Автоматизовані функції оновлення та конфігурації інструменту сприяють підтримці його актуальності з мінімальними зусиллями.

е) Спільнота та підтримка

Активна та допоміжна спільнота користувачів навколо інструменту автоматизації тестування є цінним ресурсом. Вона може сприяти швидкому вирішенню проблем, обміну знаннями та найкращими практиками. Форуми, чати, інтернет-конференції, та вебінари створюють платформу для обговорення, де користувачі можуть отримати допомогу та поради від однодумців і експертів.

Добре структуровані керівництва, посібники для швидкого старту, та докладні інструкції з налаштування та використання покращують здатність команди ефективно використовувати інструмент. Комерційна підтримка, запропонована розробниками інструменту, може бути вирішальною для компаній, які працюють над бізнес-критичними системами. Вона забезпечує гарантію своєчасної допомоги в разі виникнення проблем, доступ до професійних консультацій та індивідуальних рішень.

1.3 Висновок

Аналіз сучасних підходів DevOps та практик CI/CD підкреслює необхідність інтеграції тестування на всіх етапах розробки для забезпечення якості та оперативності виходу продуктів на ринок. Огляд популярних інструментів для автоматизації тестування, їхніх переваг, а також зростання використання методологій BDD та AI/ML у тестуванні, демонструє прогресивне рухання галузі в напрямку оптимізації та автоматизації.

2 ПОСТАНОВКА ЗАВДАННЯ

2.1 Ціль, призначення, техніко-економічне обґрунтування доцільності її вирішення та сутність задачі

Мета дослідження полягає у підвищенні ефективності процесів безперервної розробки програмного забезпечення (CI/CD) за допомогою впровадження автоматизованих тестових конвеєрів. Але мета полягає не лише у зменшенні часу розробки та забезпеченні високої якості продукту, але й у оптимізації витрат ресурсів та покращенні загальної продуктивності розробки.

Автоматизовані тестові конвеєри призначені для інтеграції та автоматизації різних видів тестувань на всіх етапах розробки програмного забезпечення. Це дозволяє підвищити точність тестувань, знизити ризик людських помилок та прискорити процес розгортання продукту.

Техніко-економічне обґрунтування полягає у впровадженні автоматизованих тестових конвеєрів у CI/CD процеси, що дозволяє значно скоротити витрати часу та ресурсів на ручне тестування, знизити кількість помилок та підвищити надійність програмного продукту. Це, в свою чергу, веде до зниження загальних витрат на розробку та підтримку продукту, а також до збільшення конкурентоспроможності та вартості кінцевого продукту на ринку.

Сутність задачі полягає у вивченні та визначенні найбільш ефективних методів та інструментів для створення автоматизованих тестових конвеєрів, інтегрованих з CI/CD процесами. Це включає аналіз поточних методів тестування, вибір оптимальних інструментів, розробку процесів та процедур, які забезпечать адаптацію до змінних умов розробки та вимог до продукту.

2.2 Обмеження та характеристики

При розробці та впровадженні автоматизованих тестових конвеєрів у процеси CI/CD, важливо враховувати специфічні обмеження та ключові характеристики, щоб забезпечити їхню ефективність та доцільність у конкретному контексті розробки. Ці параметри допомагають визначити рамки проекту та забезпечують чітке розуміння очікуваних результатів.

Таблиця 2-1 – Потенційні обмеження

Характеристика	Опис
Масштабованість	Система має бути здатна адаптуватися до зростаючої кількості тестів та розширення проекту.
Інтеграційна сумісність	Легка інтеграція з існуючими інструментами та системами, які використовуються в CI/CD процесах.
Гнучкість	Спроможність системи адаптуватися до змінних умов та специфічних вимог проекту.
Безпека	Захист даних і тестових середовищ, запобігання несанкціонованому доступу чи витоку інформації.
Ефективність	Оптимізація використання ресурсів та часу, мінімізація ручних операцій у тестуванні.

Ці характеристики відіграють критичну роль у забезпеченні того, що автоматизовані тестові конвеєри не лише відповідають технічним потребам проекту, але й сприяють загальній стратегії розвитку та ефективності розробки.

2.3 Задачі дослідження

В контексті визначеної мети дослідження, встановлення конкретних задач є ключовим для досягнення ефективного впровадження автоматизованих тестових конвеєрів у процеси CI/CD. Ці задачі визначають конкретні кроки та напрямки діяльності, необхідні для реалізації цілей проекту.

Таблиця 2-2 – Задачі дослідження

Задача	Опис задачі
Аналіз існуючих конвеєрів	Оцінка поточного стану автоматизованих тестових конвеєрів і визначення кращих практик.
Розробка критеріїв оцінки	Встановлення параметрів для оцінки ефективності, продуктивності та надійності тестових конвеєрів.
Проектування структури конвеєра	Створення детального опису

	архітектури та компонентів автоматизованого тестового конвеєра.
Інтеграція з іншими інструментами CI/CD	Визначення та реалізація методів інтеграції автоматизованого тестового конвеєра з іншими інструментами та системами CI/CD.

2.4 Вхідні та вихідні дані дослідження

Для забезпечення повноцінного аналізу та розробки автоматизованих тестових конвеєрів у CI/CD, важливо чітко визначити вхідні та вихідні дані дослідження. Ці дані формують основу для встановлення параметрів розробки, аналізу результатів та їх подальшої інтерпретації.

Таблиця 2-3 – Вхідні данні

Тип Даних	Опис Даних
Вихідний код	Вихідний код програмного забезпечення, з яким працюють тестові конвеєри.
Вимоги до ПЗ	Визначення функціональних та нефункціональних вимог до програмного продукту.
Дані проектування	Архітектурні схеми, діаграми компонентів і відомості про технічну структуру системи.

Таблиця 2-4 – Вихідні данні

Тип Даних	Опис Даних
Результати аналізу	Проведений аналіз існуючих інструментів.
Рекомендації	Напрямки для інтеграції автоматизованих тестових конвеєрів у процеси CI/CD, пропозиції з оптимізації.

Вхідні дані забезпечують фундамент для розробки та аналізу, в той час як вихідні дані представляють результати цієї роботи. Ці дані допомагають визначити ефективність впроваджених рішень, оцінити їх вплив на процеси розробки та забезпечити цінні рекомендації для подальшої оптимізації та вдосконалення.

2.5 Критерії оцінювання цілей, що мають бути реалізованими

У контексті встановленої мети дослідження - підвищення ефективності процесів безперервної розробки програмного забезпечення через впровадження автоматизованих тестових конвеєрів - важливо визначити чіткі критерії, за якими можна оцінити успішність реалізації цілей. Ці критерії допомагають виміряти прогрес та визначити ефективність запроваджених рішень.

Таблиця 2-5 – Критерії успішності реалізації цілей

Критерій	Опис Критерію
Зниження часу на розробку	Міра, наскільки використання автоматизованих тестових конвеєрів скорочує загальний час розробки.
Покращення якості продукту	Оцінка зменшення кількості помилок у продукті, покращення стабільності та надійності.
Оптимізація витрат	Аналіз ефективності використання ресурсів, зменшення витрат на ручне тестування.
Сумісність з CI/CD	Спроможність системи інтегруватися та працювати в рамках наявних CI/CD процесів.
Масштабованість	Здатність тестових конвеєрів адаптуватися та ефективно працювати при збільшенні обсягу роботи.
Зворотний зв'язок	Швидкість і якість отримання зворотного зв'язку від тестових процедур для розробників.

3 МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ

3.1 Огляд інструментів CI/CD

Перш ніж приступити до поглибленого аналізу інструментів CI/CD розглянемо більш детально найпопулярніші реалізації.

а) GitHub Actions – це інтегрована система автоматизації розробки, яка надається платформою GitHub. Цей інструмент дозволяє розробникам створювати, налаштовувати та виконувати автоматичні робочі процеси для їхніх репозиторіїв на GitHub. Історично, GitHub Actions було представлено в 2018 році, і відтоді цей інструмент значно розвинувся. GitHub Actions дозволяє вам автоматизувати тестування, збірку, розгортання та інші операції для вашого програмного коду [17].

Особливістю GitHub Actions є його вбудована інтеграція з репозиторіями на GitHub, що спрощує налаштування автоматичних процесів та спільну роботу над розробкою з командою. Цей інструмент також підтримує велику кількість різних дій, які можуть бути виконані автоматично при визначених подіях у вашому репозиторії.

б) Jenkins, спочатку відомий як Hudson, був створений у 2004 році розробником Kohsuke Kawaguchi, який працював у компанії Sun Microsystems. Він прагнув створити інструмент для виконання неперервної інтеграції, щоб тестувати код перед його залученням до основного репозиторію, уникаючи проблем із збіркою. У 2011 році, після поглинання Sun Microsystems компанією Oracle, виникла проблема з правами на назву, що призвела до зміни назви проекту з Hudson на Jenkins. Hudson і Jenkins продовжували розвиватися як два незалежні проекти, але Jenkins виявився популярнішим, і Hudson більше не розвивається [18].

Jenkins підтримує понад 1700 плагінів, які збагачують процеси інтеграції, автоматизації та доставки програмного забезпечення, надаючи можливість налаштування під конкретні потреби. Він може бути використаний для різних мов програмування і інтегрується з більшістю систем контролю версій та баз даних помилок. Jenkins Pipeline дозволяє користувачам створювати моделі для CD, використовуючи різноманітні плагіни для визначення кроків процедур від контролю версій до користувачів.

Jenkins особливо потужний, коли великі команди розробників працюють над єдиним проектом, оскільки традиційні методи можуть призвести до значної кількості конфліктних комітів коду, які вимагають складного усунення помилок.

в) GitLab був створений українським розробником Дмитром Запорожцем та голландським розробником Сітсе Сійбрандієм у 2011 році та розроблявся як відкрита платформа для спільного кодування. З 2014 року GitLab Inc. почала активно розвивати проект, використовуючи модель freemium. Компанія відома як одна з найбільших повністю віддалених компаній у світі.

GitLab CI є інструментом розробки програмного забезпечення, що дозволяє організаціям впроваджувати методології "неперервності" CI/CD. Використання GitLab CI дозволяє виявляти помилки коду та вади на ранніх стадіях циклу розробки програмного забезпечення [19].

До особливостей застосування можно віднести інтеграцією з Google Kubernetes Engine, що спрощує процес створення нового кластера для розгортання додатків. Платформа також підтримує інтеграцію з різними середовищами, такими як Docker і Kubernetes. GitLab надає можливість управління весь життєвий цикл розробки програмного забезпечення, включаючи планування, створення, перевірку, упаковку, розгортання, конфігурацію та моніторинг.

г) Azure DevOps – це інтегрована платформа для розробки програмного забезпечення, яка надає розробникам та командам інструменти для CI/CD. Цей інструмент розроблений компанією Microsoft і включає в себе низку функцій для спрощення розробки, тестування та розгортання програмного коду. Історично, Azure DevOps був відомий як Visual Studio Team Services та Team Foundation Server. Він став ключовим інструментом для розробників, які працюють з платформою Azure та іншими технологіями Microsoft.

Azure DevOps надає можливості автоматизованого тестування, збірки, розгортання та моніторингу програмного коду. Він також дозволяє командам спільно працювати над проектами, використовуючи інструменти для керування завданнями, контролю версій та спільної роботи над кодом. Azure DevOps є потужним інструментом для організацій, що прагнуть досягти високої продуктивності та якості в розробці програмного забезпечення.

д) Travis CI, заснований у 2011 році у Берліні, Німеччина, був першою службою неперервної інтеграції, яка надавала послуги відкритим проектам

безкоштовно. Travis CI – це хостингова служба неперервної інтеграції, яка використовується для збірки та тестування проектів програмного забезпечення, розміщених на GitHub. Він підтримує різні платформи та інтегрується з багатьма системами контролю версій. Travis CI налаштовується шляхом додавання файлу ".travis.yml" до кореневого каталогу репозиторію, який вказує мову програмування, бажане середовище збірки та тестування, а також різні інші параметри. Служба може бути налаштована так, щоб виконувати тести на різних машинах з різним встановленим програмним забезпеченням [20].

е) CircleCI – це хмарна платформа для неперервної інтеграції (CI) та неперервної доставки (CD), яка спрощує процес автоматизації розробки програмного забезпечення. Інструмент спрощує розробку та тестування програмного коду, допомагаючи командам розробників швидше та надійніше випускати нові версії програмного забезпечення.

CircleCI дозволяє інтегрувати різні інструменти, такі як GitHub, GitLab, та інші, для автоматизації процесу розробки. Цей інструмент також підтримує контейнеризацію, що дозволяє розробникам виконувати свої робочі процеси у віртуальних середовищах.

ж) Bitbucket Pipelines – це інтегрований інструмент CI/CD [21], який надається платформою Bitbucket, що належить компанії Atlassian. Цей інструмент дозволяє розробникам автоматизувати процеси тестування, збірки та розгортання свого програмного коду. Інтеграція з репозиторіями на Bitbucket спрощує налаштування та використання автоматичних робочих процесів. Він підтримує використання контейнерів Docker, що дозволяє розробникам створювати ізольовані середовища для тестування та розгортання.

з) AWS CodePipeline – це послуга від Amazon Web Services(AWS), яка надає можливість автоматизувати CI/CD, була запущена у 2015 році. Вона дозволяє розробникам створювати робочі потоки для автоматичного тестування, збірки та розгортання програмного коду [22].

AWS CodeStar – це інтегрована платформа розробки програмного забезпечення від AWS (з 2017 року), яка об'єднує інструменти для керування репозиторіями, автоматичного тестування, збірки та розгортання програмного коду. Вона спрощує процес CI/CD і дозволяє розробникам швидше розгортати свої додатки на AWS.

и) TeamCity представлена компанією JetBrains у 2006 році та стала однією з перших систем CI/CD на ринку. Інструмент представляє сервер неперервної

інтеграції. Він доступний як локальна установка або хмарна служба. TeamCity On-Premises може бути встановлений на Windows, Linux і macOS або працювати в Docker. TeamCity Cloud працює на Amazon AWS. Цей інструмент дозволяє автоматизувати процеси тестування та збірки програмного коду, забезпечуючи надійну та ефективну розробку програмного забезпечення, роблячи його універсальним інструментом для команд розробників [23].

Особливості TeamCity включають в себе підтримку паралельних тестів, інтеграцію з системами керування версіями, інтеграцію з Docker та можливість налаштування складних робочих потоків.

к) Google Cloud Build - це послуга надається компанією Google з 2018 року. Вона дозволяє розробникам створювати, тестувати та розгортати свій код на інфраструктурі Google Cloud Platform. Інструмент інтегрується з іншими сервісами Google Cloud, такими як Google Kubernetes Engine, Docker та інші [24].

Особливістю Google Cloud Build є можливість налаштовувати складні робочі потоки з використанням конфігураційних файлів. Це дозволяє розробникам автоматизувати всі аспекти розробки та розгортання програмного коду в хмарному середовищі Google Cloud.

л) Bamboo – ще один інструмент CI/CD від компанії Atlassian. Основна особливість Bamboo полягає в можливості створення послідовних конвеєрів для автоматизації процесу збирання, тестування та розгортання програмного коду. Він підтримує різні платформи, включаючи Linux, Windows, macOS та Solaris, але, на відміну від Bitbucket Pipelines, не інтегрован у Bitbucket.

м) JetBrains Space – це інтегрована платформа для розробки, спрямована на спрощення співпраці та автоматизацію процесів розробки. Заснована компанією JetBrains в 2019 році, вона надає інструменти для керування проектами, комунікації, CI/CD та автоматизації завдань. Space полегшує співпрацю розробників, забезпечуючи їх інструментами для зручного спілкування та спільної роботи над кодом. Він також підтримує автоматизовану інтеграцію тестів у CI/CD, що допомагає забезпечити високу якість продукту. Space є універсальною платформою, ідеально підходить для розробників, команд та підприємств будь-якого масштабу, сприяючи їм досягненню кращих результатів у розробці програмного забезпечення [25].

н) Drone – це інструмент для автоматизації CI/CD процесів, створений у 2012 році. Він дозволяє розробникам автоматизувати збирання, тестування та

розгортання програмного забезпечення. Drone підтримує інтеграцію з різними системами контролю версій і хмарними платформами. Його особливості включають простий конфігураційний файл у форматі YAML, масштабовану архітектуру, можливість використання контейнерів для збирання та виконання тестів, а також можливість налаштування різних типів спостереження за подіями. Drone допомагає командам розробників автоматизувати процеси розробки та забезпечує швидке та надійне розгортання програмного забезпечення [26].

о) AppVeyor – це інструмент автоматизації CI/CD, створений у 2011 році. Він призначений для збирання, тестування та розгортання програмного забезпечення. AppVeyor має ряд важливих функцій, включаючи підтримку для різних систем контролю версій, таких як GitHub, GitHub Enterprise, Bitbucket, GitLab, Azure Repos, Kiln, Gitea та користувацьких репозиторіїв. Надає змогу налаштовувати збірки за допомогою файлу YAML або інтерфейсу користувача. Кожна збірка виконується в ізольованому та чистому середовищі, а також має вбудований інструмент розгортання та сервер NuGet. AppVeyor підтримує збірки гілок та запитів на злиття, що допомагає процесі розробки ПО. Крім того, надає професійну підтримку та можливість взаємодії з активною спільнотою користувачів [27].

п) GoCD – це інструмент автоматизації CI/CD, випущений у 2007 році. Він надає змогу створювати складні робочі процеси та моделювати CD-робочі потоки з паралельним виконанням та управлінням залежностями. Завдяки інтерактивній карті потоку значення, GoCD дозволяє візуалізувати шлях до випуску в одному вигляді, виявляти неефективності та оптимізувати процес. Інтеграція з популярними хмарними середовищами, такими як Kubernetes, Docker, AWS і багатьма іншими, робить GoCD ідеальним вибором для хмарних розгортань. Інструмент також пропонує розширений механізм слідування, який дозволяє відстежувати кожну зміну в реальному часі від фіксації до розгортання. GoCD має розширену систему плагінів, яка інтегрується з багатьма зовнішніми інструментами та сервісами для спрощення робочого процесу [28].

р) CodeShip – це сервер для CI, який був випущений у 2011 році. Він надає можливість конфігурувати збірки як в YAML, так і через користувацький інтерфейс. Codeship забезпечує ізольоване, чисте середовище для кожної збірки і включає в себе вбудований сервер NuGet. Інструмент підтримує збірки гілок

та запитів на злиття. Загалом, Codeship пропонує зручну інтеграцію з різними репозиторіями та відмінно підходить для автоматизації процесу поставки програмного забезпечення.

3.2 Поглиблений аналіз інструментів CI/CD

Важливим аспектом впровадження автоматизованих тестових конвеєрів є вибір відповідних інструментів CI/CD, які забезпечують необхідну інтеграцію, продуктивність та гнучкість. Таблиця 3-1 нижче представляє порівняльний огляд ключових характеристик цих інструментів.

Таблиця 3-1 – Список популярних інструментів CI

	Відкритий код	Хостинг	Безкоштовна версія	Вартість ліцензії агента для збирання	Підтримувані платформи
GitHub Actions	Ні	Хмара	Так	Включені виконавчі одиниці для кожного рівня.	Linux, Windows і macOS
Jenkins	Так	Самостійно	Так	Безкоштовно	Linux, Windows і macOS
GitLab CI	Ні	Хмара і самостійно	Так	Включені одиниці збирання для кожного рівня. Додаткові одиниці для запуску збірок на спільних конвеєрах починаються з $10 за 1000 хвилин.	Linux, Windows, macOS і Docker

Azure DevOps	Ні	Хмара і самостійно	Так	1 конвеєр включено безкоштовно. Додаткові конвеєр починаються з $15 на місяць (самостійне використання) або $40 на місяць (хмара).	Linux, Windows і macOS
CircleCI	Ні	Хмара і самостійно	Так	Хвилини збирання включено для кожного рівня. Кредити можуть бути обмінювані на хвилини збирання, користувачів та додаткову мережу та сховище.	Linux, macOS, Windows, GPU, ARM і Docker
Travis CI	Ні	Хмара і самостійно	Ні	Обмеження на одночасні завдання залежить від рівня. Безліч хвилин збирання на будь-якому рівні.	Linux, macOS і iOS
Bitbucket Pipelines	Ні	Хмара	Так	Хвилини збирання включено для кожного рівня. Додаткові хвилини починаються з $10 на місяць за 1000 хвилин.	Linux, Windows і macOS

AWS CodePipeline / AWS CodeStar	Ні	Хмара	Так	Ціноутворення за конвеєр. Зберігання в AWS обтяжує додаткові витрати.	Linux, Windows і macOS
TeamCity	Ні	Хмара і самостійно	Так	Самостійне господарювання: 3 агента для збирання та необмежене збирання включено у безкоштовному рівні. Додаткові агенти починаються з $239 на рік. Хмара: від $45 на місяць для 3 комітентів.	Linux, Windows, macOS і Docker
Google Cloud Build	Ні	Хмара	Так	Серверний платформа зі сплатою за хвилину збирання.	Docker
Bamboo	Ні	Самостійно	Так	1 віддалений агент включено в базову ціну. Ціноутворення на 5 агентів починається з $640 на агента на рік.	Linux, Windows, macOS і Solaris

JetBrains Space	Ні	Хмара і самостійно	Так	Хмара: 2000 хвилин збирання включено в безкоштовний план. Платні плани починаються з 4000 хвилин та дозволяють придбати додаткові ресурси – $8 на місяць за 1000 хвилин. Самостійне господарювання: 3 одночасних робочих робітника доступні на безкоштовному плані, а з 10 – на платних планах.	Linux, Windows і macOS
Drone	Ні	Хмара і самостійно	Так	Недоступно	Linux, Windows і ARM
AppVeyor	Ні	Хмара і самостійно	Так	Максимум 2 одночасні завдання на платному плані.	Linux, Windows і macOS
GoCD	Так	Самостійно	Так	Безкоштовно	Linux, Windows і macOS
Codeship	Ні	Хмара	Ні	$75 на місяць за кожну одночасну збірку з CodeShip Enterprise.	Linux, Windows і macOS

3.2.1 Аналіз характеристик інструментів

Вибір інструменту CI/CD є важливим завданням для розробників і організацій, оскільки він впливає на швидкість, надійність та ефективність процесу розробки програмного забезпечення.

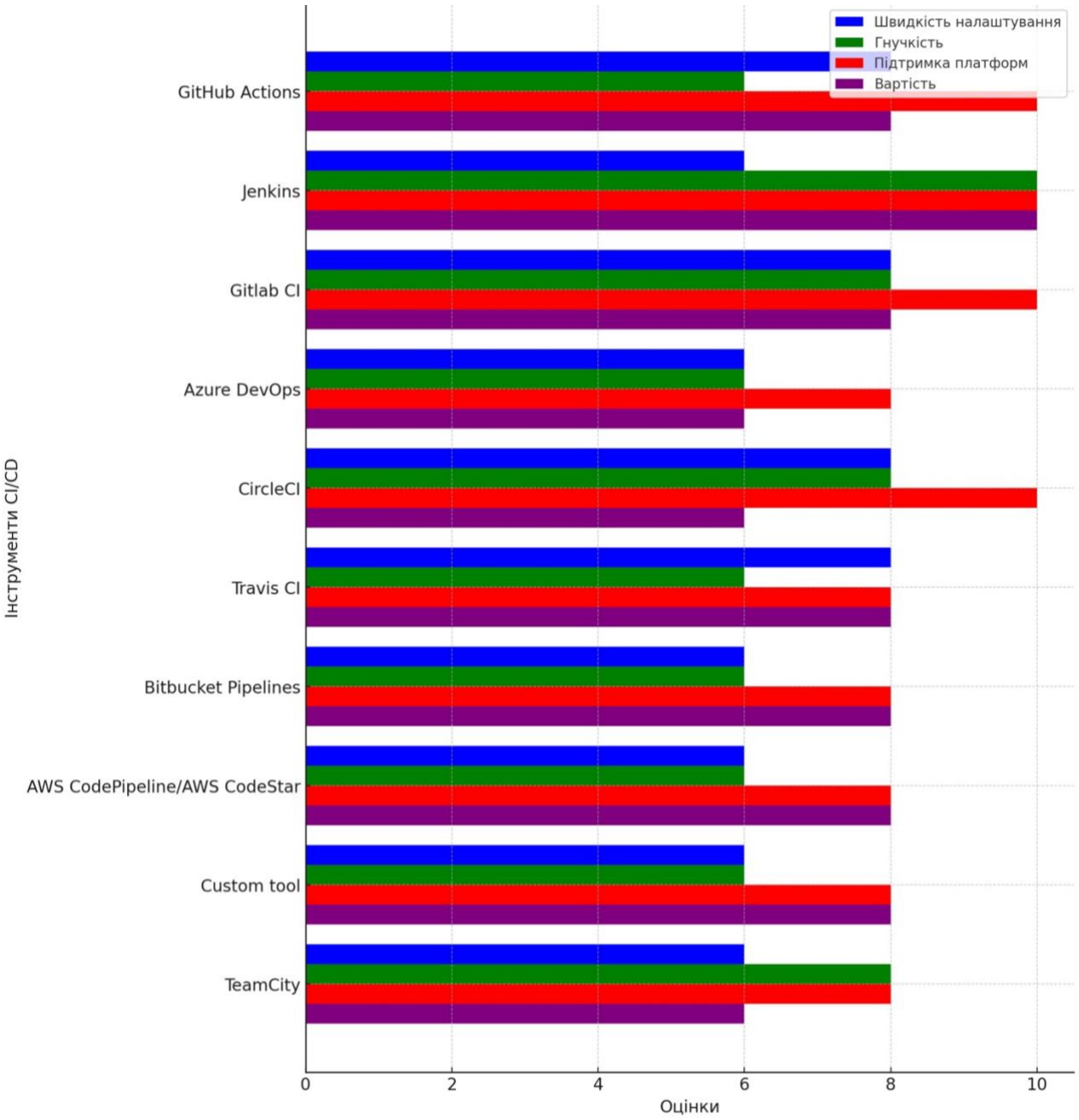

Рисунок 3-1 – Порівняльний аналіз інструментів CI/CD

На основі представленої таблиці 3-1, виконано аналіз ключових аспектів інструментів CI/CD, таких як відкритий код, хостинг, наявність безкоштовних

версій, вартість ліцензії та підтримувані платформи. Це дозволяє зрозуміти, які інструменти найкраще відповідають специфічним потребам різних проектів та середовищ розробки.

а) Вибір між інструментами з відкритим кодом, такими як Jenkins, і хмарними рішеннями, наприклад, GitHub Actions, залежить від потреб у персоналізації та інтеграції з іншими інструментами. Хмарні рішення часто пропонують більшу гнучкість і простоту в налаштуванні.

б) Багато інструментів пропонують безкоштовні версії з обмеженим функціоналом, що є важливим для невеликих проектів або команд з обмеженим бюджетом. Однак, для більш складних потреб, платні версії надають додаткові можливості та підтримку.

в) Різноманітність підтримуваних платформ є ключовим фактором для вибору інструменту, особливо в середовищах, де використовуються різні операційні системи та технології.

З урахуванням аналізу, можна сформулювати основні критерії для вибору інструменту CI/CD:

– Підтримка поточних технологічній. Інструмент повинен підтримувати систему керування версіями (VCS), мови програмування та платформи збірки та тестування, що використовуються в проекті.

– Можливість майбутньої міграції. Важливо передбачати можливість майбутнього переходу до інших технологій або розширення функціональності. Інструмент повинен дозволяти легко адаптуватися до змін в екосистемі проекту.

– Свобода налаштування конвеєрів. Інструмент повинен надавати можливість налаштовувати CI/CD конвеєри відповідно до потреб процесу розробки. Наявність CI/CD як коду дозволяє легко покращувати та налагоджувати визначення конвеєрів.

– Індивідуальний підхід, який повинен відображати унікальні вимоги проекту. Наприклад, для великих монорепозиторіїв може бути важливо вибирати інструмент, який дозволяє активувати CI/CD тільки для певних директорій.

– Підтримка систем керування версіями.

– Свобода налаштування. Інструменти CI/CD як код надають можливість налаштовувати конвеєри за допомогою програмування, що полегшує розробку та налагодження визначень збирання та тестування.

Також при виборі CI/CD інструменту можна керуватися дослідницьким підходом, який передбачає аналіз інших експертиз і рекомендацій при виборі CI/CD інструменту. Інші розробники та організації можуть поділитися власними враженнями та кращими практиками.

Так деякі організації вибирають розробку власних CI-інструментів з ряду ключових причин. По-перше, це пов'язано з необхідністю задоволення специфічних потреб, процесів та робочих потоків команди. Власний CI-інструмент надає можливість створити рішення, яке ідеально відповідає цим вимогам.

По-друге, інтеграція є важливою складовою обраного підходу. Власний інструмент може бути краще інтегрованим в складні системи або відповідати специфічним вимогам організації, що полегшує їх використання та підтримку.

По-третє, само розроблений інструмент може надавати специфічний функціонал, який важко знайти в загальнодоступних інструментах. Це дає можливість враховувати унікальні вимоги та процеси організації, що сприяє покращенню продуктивності та якості розробки.

3.2.2 Застосування критеріїв до аналізу інструментів CI/CD

Зниження часу на розробку

– GitHub Actions і GitLab CI зазвичай пропонують швидке налаштування та інтеграцію, що може значно скоротити час розробки.

– Jenkins забезпечує гнучкість, але може вимагати більше часу на налаштування.

Покращення якості продукту

– CircleCI і Travis CI надають надійні тестові можливості для підвищення стабільності продукту.

– Azure DevOps пропонує комплексні інструменти для моніторингу якості.

Оптимізація витрат

– Jenkins є відмінним вибором для обмежених бюджетів.

– TeamCity та Bamboo можуть бути більш коштовними, але пропонують більш складні рішення.

Сумісність з CI/CD

– GitLab CI та GitHub Actions ідеально підходять для інтеграції з відповідними екосистемами.

– Azure DevOps та AWS CodePipeline забезпечують глибоку інтеграцію з продуктами Microsoft та AWS.

Масштабованість

– CircleCI та Azure DevOps підходять для великих проектів з високими вимогами до масштабованості.

– Travis CI та AppVeyor краще підходять для середніх та малих проектів.

Зворотний зв'язок

– GitLab CI та GitHub Actions забезпечують швидкий зворотний зв'язок через інтеграцію з системами керування версіями.

– Jenkins дозволяє гнучко налаштовувати зворотний зв'язок, але потребує додаткових налаштувань.

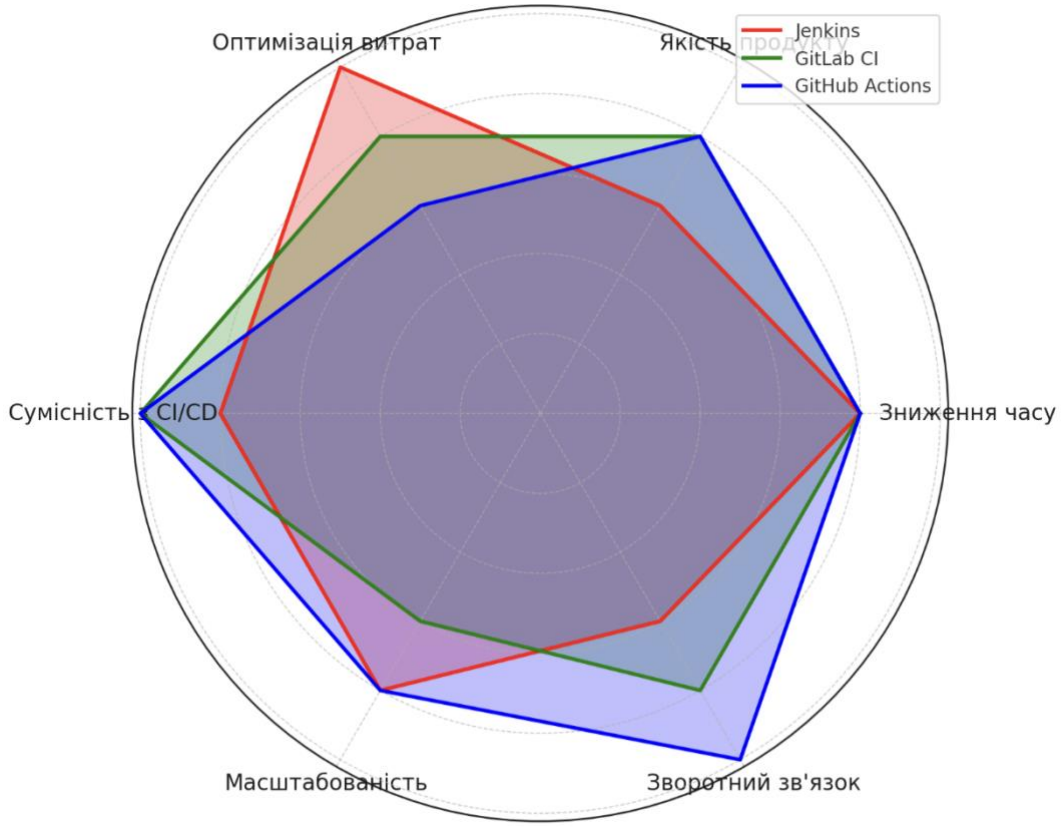

Рисунок 3-2 – Порівняльний аналіз Jenkins, GitLab CI, та GitHub Actions

Порівняльний аналіз вартості різних інструментів CI/CD, залежно від обсягу їх використання, включаючи індивідуальне використання, малий бізнес та великі підприємства виявляє як фінансову доступність, так і потенційні фінансові зобов'язання, пов'язані з кожним інструментом.

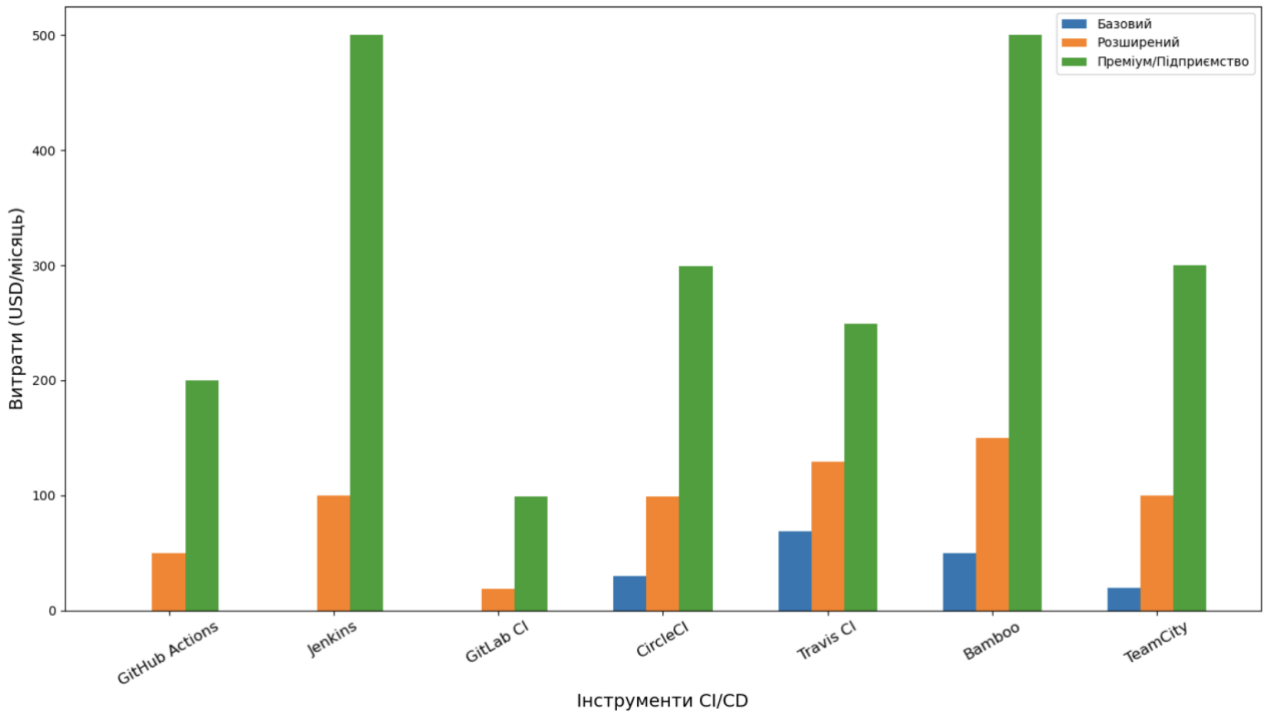

Рисунок 3-3 – Потенційні фінансові витрати

Для індивідуального використання, витрати були визначені як [0, 0, 0, 30, 69, 50, 20] USD відповідно для інструментів GitHub Actions, Jenkins, GitLab CI, CircleCI, Travis CI, Bamboo, TeamCity. Це відображає, що деякі інструменти, як-от GitHub Actions, Jenkins та GitLab CI, пропонують безкоштовні плани, що є ідеальними для індивідуальних розробників або малих проектів, де бюджет є обмеженим. Інші інструменти, такі як CircleCI та Travis CI, хоча й пропонують платні плани, їх вартість залишається порівняно низькою, що робить їх доступними для індивідуального використання.

Для малого бізнесу, витрати складають [50, 100, 19, 99, 129, 150, 100] USD для тих самих інструментів. Ці значення відображають зростання витрат зі збільшенням розміру проектів та потреб команд. Малі підприємства часто потребують більш складних функцій та підтримки, що призводить до вищих витрат.

Для великих підприємств, вартість значно зростає і складає [200, 500, 99, 299, 249, 500, 300] USD. Це відображає потребу у великих масштабах масштабування, високому рівні безпеки, складних інтеграціях та професійній підтримці, які є критичними для великих організацій. Такі інструменти, як Jenkins та TeamCity, які пропонують високий рівень налаштувань та масштабування, можуть вимагати значних інвестицій в інфраструктуру та підтримку для великих підприємств.

Ці визначення витрат відображають не тільки фінансові аспекти вибору інструментів CI/CD, але й підкреслюють важливість вибору інструменту, який найкраще відповідає потребам користувача, залежно від розміру та масштабу його проектів або організації.

3.2.3 Кейс-стаді або приклади з практики

Після детального аналізу інструментів CI/CD, наступним кроком є розгляд реальних випадків їх використання. Це допоможе нам краще зрозуміти, як теоретичні переваги та недоліки відображаються у практичному застосуванні. Ми зосередимося на кейс-стаді, які ілюструють використання Jenkins, GitLab CI та GitHub Actions в реальних проектах, їх вплив на процеси розробки та отримані результати [29,30,31].

Кейс 1: Jenkins у Стартапі

– Ситуація: Фінтех-стартап потребує гнучкого рішення для автоматизації розробки.

– Рішення: Використання Jenkins забезпечило гнучкість завдяки відкритому коду та розширеному спектру плагінів.

– Результат: Суттєве зниження часу на розробку, підвищення гнучкості в CI/CD процесах.

Кейс 2: GitLab CI в Середньому Бізнесі

– Ситуація: Компанія середнього розміру шукає інтегроване рішення для управління проектами.

– Рішення: Вибір GitLab CI через тісну інтеграцію з системою керування версіями та зручність використання.

– Результат: Підвищення ефективності роботи команди, поліпшення якості продукту, оптимізація процесів тестування та розробки.

Кейс 3: GitHub Actions у Великій Організації

– Ситуація: Організація шукає рішення для автоматизації робочих процесів та CI/CD.

– Рішення: Впровадження GitHub Actions для автоматизації звітів про безпеку, регресійного тестування, та підтримки процесу релізу.

– Результат: Ефективність автоматизації робочих процесів, зниження ручної роботи, підвищення швидкості розробки.

3.3 Особливості інтеграції автоматичних тестів

В сучасному світі програмування, інтеграція автоматичних тестів у процес розробки програмного забезпечення є ключовим елементом для досягнення високої продуктивності та якості продукції. Цей процес, відомий як Continuous Testing (CT), є частиною більш широкої практики CI/CD. Основна мета CT – забезпечити регулярне та автоматизоване виконання тестів, щоб швидко виявляти помилки та несправності.

Основою ефективного автоматичного тестування є його інтеграція з CI/CD конвеєрами. Ця інтеграція передбачає, що тести автоматично запускаються при кожному коміті коду в репозиторій. Це дозволяє розробникам швидко виявляти та виправляти помилки, що, в свою чергу, зменшує час, необхідний для розгортання нових функцій або виправлень.

Вибір правильних інструментів для автоматичних тестів є критично важливим. Популярні інструменти, такі як Selenium, JUnit, та Jenkins, пропонують широкі можливості для автоматизації тестування. Вибір залежить від мови програмування, складності проекту, та інших технічних і організаційних чинників.

Розробка стратегії автоматичного тестування вимагає врахування різних аспектів: які тести слід автоматизувати, як часто їх слід запускати, та як інтегрувати тестування в загальний процес розробки. Важливо забезпечити, щоб автоматизовані тести були надійними, швидкими та легко підтримуваними.

Автоматизація тестування стикається з декількома ключовими викликами. Одним з основних є підтримання актуальності тестових сценаріїв у відповідності до постійно змінюваного коду. Це вимагає регулярного

оновлення та оптимізації тестів для забезпечення їхньої ефективності та релевантності. Інший важливий аспект – це вибір між швидкістю виконання тестів та їхньою всебічністю.

Не всі тести потребують автоматизації. Важливо ідентифікувати ті, які мають найбільшу цінність та частоту використання. Як правило, перевага надається модульним тестам, інтеграційним тестам та тестам регресії. Тести користувацького інтерфейсу, які часто вимагають значних зусиль для підтримки, автоматизуються з обережністю.

Ефективна інтеграція автоматичних тестів також залежить від їх здатності інтегруватися з іншими інструментами, використовуваними в процесі розробки. Це включає системи контролю версій, інструменти для збірки проектів, а також системи моніторингу та журналювання. Така інтеграція дозволяє швидко реагувати на помилки, виявлені під час тестування, та підтримувати високий рівень якості коду.

В складних системах, де розробка ведеться одночасно у кількох модулях або компонентах, автоматизація тестування стає особливо важливою. Тут використовується підхід, що дозволяє одночасно тестувати різні частини системи та швидко ідентифікувати залежності та конфлікти між ними. Особливу увагу слід приділяти тестам інтеграції та тестам на відповідність інтерфейсам.

Оптимізація процесу автоматичного тестування вимагає розуміння та балансу між швидкістю, вартістю та якістю. Важливо розробити стратегію, яка мінімізує час виконання тестів, забезпечуючи при цьому їхню високу надійність та покриття коду. Паралельне виконання тестів, кешування результатів, та використання "розумних" алгоритмів для визначення, які тести потрібно запускати, можуть значно покращити ефективність процесу.

Сучасні технології, такі як контейнеризація та віртуалізація, надають значні переваги для автоматичного тестування. Використання Docker, Kubernetes та інших подібних технологій дозволяє створювати ізольоване та уніфіковане середовище для тестування, що підвищує його стабільність та відтворюваність.

Ефективна звітність та аналіз результатів тестування є критично важливими для виявлення тенденцій, зон ризику та можливостей для оптимізації. Інструменти, такі як SonarQube, можуть автоматично аналізувати

якість коду та відображати деталізовані звіти про результати тестів, дозволяючи командам швидко реагувати на проблеми.

Як вже було згадано у першому розділі розвиток технологій штучного інтелекту (AI) та машинного навчання (ML) відкриває нові можливості для автоматизації тестування. AI та ML можуть допомагати у визначенні складних взаємодій в коді, прогнозуванні потенційних точок збою, та навіть автоматичному генеруванні тестових сценаріїв.

Вивчення практичних кейс-стаді дозволяє глибше зрозуміти, як автоматизація тестування застосовується в реальних проектах. Наприклад, в проектах розробки великих веб-додатків, де використовується мікросервісна архітектура, автоматизовані тести відіграють ключову роль у забезпеченні безперервності та стабільності сервісів. Ці тести часто включають широкий спектр перевірок - від юніт-тестування до комплексних інтеграційних та навантажувальних тестів.

Інтеграція автоматичного тестування є особливо важливою у контексті Agile та DevOps практик. В Agile-середовищах, швидкість впровадження змін та адаптивність є ключовими, і тут автоматизоване тестування допомагає підтримувати високий темп розвитку без втрати якості. У DevOps, зосередження на автоматизації та безперервному зворотному зв'язку, автоматичні тести відіграють центральну роль у забезпеченні стабільності та надійності в розгортанні продукції.

3.3.1 Кейс-стаді з використання автоматизації тестування

Наступні приклади демонструють різноманітність застосувань та значні переваги автоматизованого тестування, від розробки мобільних додатків до великомасштабних промислових і туристичних систем. Автоматизоване тестування не тільки прискорює процес тестування, але й підвищує покриття, поліпшує якість продукту та забезпечує швидше впровадження програмних рішень.

а) Автоматизація Тестування Мобільних Додатків. Розробка мобільних додатків стикається з унікальними викликами через проблеми сумісності на різних операційних системах, як-от iOS та Android. Автоматизоване тестування відіграє ключову роль у забезпеченні безперебійної функціональності додатків

на різних платформах. Воно включає створення автоматизованих тестових сценаріїв, які імітують взаємодію користувачів, наприклад, натискання кнопок або введення інформації, для перевірки функціональності та сумісності додатків. Такий підхід не тільки ефективний за часом, але й мінімізує помилки порівняно з ручним тестуванням. Автоматизоване функціональне та регресійне тестування є важливими в розробці мобільних додатків, забезпечуючи стабільність та продуктивність додатків на широкому спектрі пристроїв та версій ОС [32].

б) Використання автоматизації тестування «без коду» компанією KUKA. KUKA AG, компанія, що спеціалізується на промислових роботах та автоматизованих системах, зіткнулася з викликами через ручне тестування при масштабуванні своєї інженерної групи. Вони впровадили «безкодову» автоматизацію тестування за допомогою Tricentis Tosca, що призвело до значних поліпшень. Вони змогли почати тестування раніше в процесі розробки, розширити покриття тестування з 20% до 80%, скоротити час тестування на 95% і прискорити цикл випуску з 6 місяців до 6 тижнів [33].

в) Впровадження AI-керованої автоматизації в Travelopia. Travelopia, глобальна туристична група, яка управляє понад 50 брендами, потребувала прискорення та покращення тестування більш ніж 200 систем через складний технологічний відбиток. Після впровадження автоматизованого тестування з підтримкою штучного інтелекту (ШІ) з AutonomIQ, досягнули значного прогресу: на 50% швидше створення тестів, на 90% зниження зусиль з підтримки тестів, на 20% збільшення покриття тестування, та в 10 разів швидше створення тестів у термінових сценаріях [33].

4 ПРОЕКТУВАННЯ ТЕСТОВОГО КОНВЕЄРА

4.1 Загальна структура конвеєра з використанням Jenkins

Загальну структуру розглядаємо на прикладі конвеєра з використанням Jenkins, що є важливим для забезпечення ефективності та надійності процесу CI/CD веб та мобільних додатків.

Центральна роль у системі належить центральному серверу Jenkins, який координує всі основні процеси CI/CD. Сервер не тільки управляє процесом збирання та тестування коду, але й інтегрується з іншими інструментами, що сприяє злагодженій роботі всієї системи.

Jenkins Job – це автоматизований процес або завдання, яке використовується в системі безперервної інтеграції Jenkins для виконання різних операцій, таких як збирання, тестування, розгортання або інші дії в розробці програмного забезпечення.

Вузли-агенти Jenkins виконують завдання збирання та тестування. Вони можуть бути спеціалізовані під конкретні задачі та середовища, наприклад, під різні версії операційних систем, які використовуються для мобільних додатків.

Системи керування версіями (VCS), такі як Git, забезпечують контроль за змінами в коді, що дозволяє здійснювати швидку інтеграцію нових змін та автоматичний запуск процесів CI/CD у відповідь на кожен коміт у репозиторій.

Інструменти автоматизованого тестування дозволяють автоматизувати процес перевірки коду, наприклад, Selenium для веб-додатків та Appium для мобільних додатків, що є ключовим для забезпечення високої якості продукту.

Інструменти звітності та моніторингу відповідають за збір даних про процеси тестування та збирання, а також їх візуалізацію і представлення в зручній формі для аналізу та ухвалення рішень.

Нарешті, Плагіни Jenkins розширюють функціональність системи, дозволяючи інтегрувати додаткові сервіси та інструменти, необхідні для специфічних потреб веб та мобільних додатків.

Використання такої структури конвеєра дозволяє оптимізувати робочі процеси розробки, забезпечуючи швидке виявлення та виправлення помилок, автоматизацію рутинних задач і підвищення продуктивності команди розробників, на рисунку А1.

4.2 Детальний опис Jenkins Pipeline

Jenkins, по суті, є двигуном автоматизації, який підтримує багато шаблонів автоматизації. Pipeline додає потужний набір інструментів автоматизації до Jenkins, підтримуючи випадки використання, які варіюються від простої безперервної інтеграції до всеосяжних CD конвеєрів. Моделюючи серію пов'язаних завдань, користувачі можуть скористатися багатьма перевагами Pipeline [34]:

а) Код: Pipeline реалізовані у коді та, як правило, перевіряються у систему контролю версій, даючи командам можливість редагувати, переглядати та ітерувати свій процес доставки.

б) Стійкість: Pipeline можуть вижити як заплановані, так і непередбачені перезапуски контролера Jenkins.

в) Пауза: Pipeline можуть необов'язково зупинятися та чекати на людський ввід або схвалення перед продовженням виконання Pipeline.

г) Універсальність: Pipeline підтримують складні реальні вимоги CD, включаючи можливість розгалуження/об'єднання, циклічність та виконання роботи паралельно.

д) Розширюваність: Плагін Pipeline підтримує користувацькі розширення до свого DSL та кілька варіантів інтеграції з іншими плагінами.

Jenkins Pipeline є ключовою концепцією, яка забезпечує набір інструментів для реалізації конвеєра. Він дозволяє визначати весь процес збирання, тестування та доставки додатків через код, який називається Jenkinsfile, на рисунку А.2. Цей файл розміщується в репозиторії проекту і описує кроки, необхідні для виконання різних завдань у рамках процесу збірки.

Нижче наведено високорівневий огляд цих концепцій:

а) Pipeline – є користувацько визначеною моделлю процесу безперервної доставки. Код Pipeline визначає весь процес збирання, який зазвичай включає етапи збирання додатка, його тестування, та доставки.

б) Node – це машина, яка є частиною середовища Jenkins і здатна виконувати Pipeline. Вона відіграє ключову роль у виконанні скриптованого синтаксису Pipeline.

в) Stage визначає концептуально відмінну підмножину завдань, які виконуються на протязі всього Pipeline. Наприклад, це можуть бути етапи build, test, deploy. Багато плагінів використовують етапи для візуалізації або представлення статусу/прогресу Jenkins Pipeline.

г) Step – це окреме завдання. Загалом, крок вказує Jenkins, що робити в конкретний момент часу або на певному етапі процесу. Наприклад, для виконання команди оболонки «make», використовується крок «sh: sh 'make'».

д) Синтаксис Pipeline. Існують декларативний та скриптований Pipeline синтаксисі. Обидва включають етапи та кроки але конструктивно різні.

– У декларативному синтаксисі Pipeline, блок pipeline визначає всю роботу, що виконується протягом усього Pipeline. Це включає визначення агентів та етапів, та виконання певних кроків на кожному з цих етапів. Також слід зауважити, що це більш нова особливість Jenkins Pipeline і вона надає багатші синтаксичні можливості порівняно зі скриптованим синтаксисом Pipeline, що дає змогу писати та читати код Pipeline легше.

– У скриптовому синтаксисі Pipeline, один або декілька блоків node виконують основну роботу на протязі всього Pipeline. Використання блоку node дозволяє розподіляти завдання для виконання, створюючи робочий простір для роботи з файлами, отриманими з системи контролю версій.

Для декларативного синтаксису можна виділити наступні складові:

1. Виконати цей Pipeline або будь-який з його етапів на будь-якому доступному агенті.
2. Визначає Stage "Build".
3. Виконати деякі кроки, пов'язані з Stage "Build".
4. Визначає Stage "Test".
5. Виконати деякі кроки, пов'язані з Stage "Test".
6. Визначає Stage "Deploy". Виконати деякі кроки, пов'язані з Stage " Deploy ".

Для скриптового синтаксису можна виділити наступні складові:

1. Виконати цей Pipeline або будь-який з його етапів на будь-якому доступному агенті.
2. Визначає Stage " Build ". Блоки Stage є необов'язковими у синтаксисі скриптового Pipeline. Однак, впровадження блоків Stage у скриптовому Pipeline

забезпечує чіткішу візуалізацію підмножини завдань/кроків кожного етапу в інтерфейсі користувача Jenkins.

3. Виконати деякі кроки, пов'язані з Stage "Build".
4. Визначає Stage "Test".
5. Виконати деякі кроки, пов'язані з Stage "Test".
6. Визначає Stage "Deploy".
7. Виконати деякі кроки, пов'язані з Stage "Deploy".

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any (1)
    stages {
        stage('Build') { (2)
            steps {
                // (3)
            }
        }
        stage('Test') { (4)
            steps {
                // (5)
            }
        }
        stage('Deploy') { (6)
            steps {
                // (7)
            }
        }
    }
}
```

Jenkinsfile (Scripted Pipeline)

```
node { (1)
    stage('Build') { (2)
        // (3)
    }
    stage('Test') { (4)
        // (5)
    }
    stage('Deploy') { (6)
        // (7)
    }
}
```

Рисунок 4-1 – Синтаксис Jenkins Pipeline

Враховуючи розглянуті особливості Jenkins можна розробити наступні рекомендації налаштування конвеєру, на рисунок А.3:

а) Установка та Конфігурація Jenkins:

– Установка Jenkins на сервер. Можна завантажити його з офіційного сайту Jenkins.

– Налаштування базові параметри Jenkins, такі як порт сервера, безпеку та доступ до системи.

б) Встановлення необхідних плагінів:

– Установка плагінів, які потрібні для процесу автоматичного тестування. Наприклад для Java веб додатку можуть знадобитися плагіни, такі як Maven Integration, Git, Allure Jenkins Plugin тощо.

в) Створення Jenkins Job:

– Створення нової Jenkins Job. Можна використовувати "Freestyle project" для простоти або "Pipeline" для більш складних процесів.

г) Налаштування джерела коду (наприклад, Git) та скрипт збірки (наприклад, Maven).

д) Інтеграція з системою контролю версій:

– Налаштування інтеграції з Git або іншою системою контролю версій, щоб Jenkins міг автоматично отримувати останні зміни коду.

е) Налаштування сценаріїв тестування:

– Опис та налаштування сценаріїв тестування в Maven (або аналогічному інструменті), які Jenkins буде виконувати.

– Використовування pom.xml для конфігурації залежностей та плагінів тестування.

ж) Автоматизація запуску тестів:

– Використовування тригерів в Jenkins для автоматичного запуску тестів після кожної зміни коду або за певним графіком.

з) Збір та аналіз результатів тестування:

– Налаштування Jenkins для збору та відображення результатів тестів, наприклад, з використанням Allure або іншого інструменту звітності.

и) Моніторинг та оповіщення:

– Установка моніторингу статусу роботи та налаштування оповіщення (наприклад, через електронну пошту або Slack), щоб команда була в курсі успіхів чи невдач тестування.

4.3 Потенційні сценарії та кращі практики використання

У процесі розробки програмного забезпечення важливо не лише вибрати відповідні інструменти для створення CI/CD конвеєра, але й розуміти, яким чином цей конвеєр може бути адаптований та впроваджений у різноманітні проекти. Розроблений тестовий конвеєр на базі Jenkins з його гнучкістю та масштабованістю може бути застосований у широкому спектрі сценаріїв - від

стартапів до корпоративних додатків, що охоплюють як веб, так і мобільні платформи.

Нижче представлені сценарії використання, які ілюструють потенційне застосування розробленої структури конвеєра в різних умовах та середовищах розробки. Кожен сценарій враховує унікальні потреби та вимоги, що висуваються до процесів автоматизації, тестування та розгортання, відповідно до специфіки проектів.

Сценарій 1: Веб-додаток стартапу

Контекст: Стартап у галузі електронної комерції розробляє веб-додаток з використанням мікросервісної архітектури. Швидкість виходу на ринок є критично важливою.

Впровадження конвеєра:

– Використання Jenkins для автоматизації тестування та розгортання кожного мікросервісу окремо.

– Налаштування пайплайнів для різних середовищ (розробка, тестування, випуск).

– Автоматичне виявлення помилок та їх виправлення перед розгортанням у робоче середовище.

Результат: Прискорення процесу доставки нових функцій та зниження ризиків помилок у робочому середовищі.

Сценарій 2: Корпоративний мобільний додаток

Контекст: Велика фінансова корпорація розробляє мобільний додаток для своїх клієнтів. Безпека та стабільність є пріоритетами.

Впровадження конвеєра:

– Використання Jenkins з плагінами для роботи з мобільними ОС (iOS, Android).

– Автоматизація регресійного тестування та перевірки на вразливості.

– Розгортання оновлень додатку через автоматизовані пайплайни після всебічного тестування.

Результат: Підвищення безпеки мобільного додатку та забезпечення його надійної роботи.

Сценарій 3: Продукт з відкритим кодом

Контекст: Команда розробників створює продукт з відкритим кодом, який регулярно оновлюється завдяки співпраці зі спільнотою.

Впровадження конвеєра:

– Налаштування Jenkins для автоматичного тестування pull-запитів від спільноти.

– Автоматизація злиття перевірених змін та їх розгортання у базові вітки.

– Використання звітів про тести для швидкого виявлення та вирішення проблем.

Результат: Збільшення швидкості впровадження змін та поліпшення якості коду завдяки тісній співпраці зі спільнотою розробників.

Аналізуючи реальні приклади можна виділити наступні кращі практики використання:

а) Автоматизація Jobs. Використання організаційних папок та багатогілкових (multibranch) Pipeline для автоматичного створення, оновлення та видалення робіт у Jenkins в залежності від репозиторіїв та гілок, що додаються чи видаляються.

б) Управління Jobs Jenkins. Оптимізація визначення Jobs Jenkins для поліпшення взаємодії користувачів та продуктивності. Наприклад, використання графіків та звітів для аналізу статусу проекту, збірка на агентах для підвищення безпеки та масштабованості, відображення помилок правильним особам через налаштовані сповіщення.

в) Інтеграція з VCS. Налаштування Jenkins для інтеграції з системами керування версіями, щоб автоматизувати збирання, тестування та розгортання відповідно до змін у коді.

г) Горизонтальне масштабування Jenkins. Встановлення кількох екземплярів Jenkins, які можуть спілкуватися один з одним, дозволяючи масштабувати Jenkins горизонтально для обробки збільшеного обсягу роботи.

д) Безпечне управління повноваженнями. Jenkins підтримує різні типи повноважень, такі як SSH-ключі, імена користувачів і паролі, API-токени. Використання кращих практик для управління повноваженнями, включаючи зберігання їх у зашифрованому форматі та обмеження доступу до них, забезпечує безпеку.

е) Використання шаблонів робіт (Job Templates). Job Templates – це попередньо налаштовані Jobs, які можуть бути використані як вихідна точка для нових робіт. Застосування Job Templates дозволяє забезпечити

послідовність у структурі та конфігурації робіт, що допомагає знизити ризик помилок та полегшити управління Jenkins Job в масштабі.

ж) Використання параметризованих збірок. Параметризовані збірки дозволяють визначати користувацькі параметри для збірок, що допомагає автоматизувати та налаштовувати процеси збирання, наприклад, вказуючи різні цілі збирання чи конфігурації. Це також полегшує повторне використання збірок для різних цілей.

з) Використання візуалізацій pipeline. Jenkins пропонує потужний інструмент візуалізації pipeline, який дозволяє відразу бачити прогрес ваших збірок. Візуалізації pipeline дозволяють швидко виявити "вузькі місця" та помилки у вашому pipeline та додати необхідні поліпшення.

и) Рольовий контроль доступу. Рольовий контроль доступу (rbac) дозволяє контролювати доступ до вашого екземпляра Jenkins на основі ролей та дозволів користувачів. Rbac допомагає забезпечити, що лише уповноважені користувачі можуть вносити зміни у вашу систему, що допомагає запобігати порушенням безпеки та випадковим змінам.

к) Використання шаблонів плагінів. При використанні плагінів у Jenkins важливо використовувати версійні плагіни, щоб забезпечити стабільність та консистентність вашого екземпляра. Версійні плагіни забезпечують надійний спосіб управління залежностями плагінів та уникнення проблем сумісності.

л) Використання міток агентів для кращого розподілу робіт при використанні вузлів-агентів важливо використовувати мітки агентів для кращого розподілу робіт на вузли з відповідними ресурсами. За допомогою міток агентів можна забезпечити, що роботи виконуються на вузлах з правильними можливостями, що може покращити продуктивність та скоротити час збирання.

м) Автоматизоване тестування. Автоматизоване тестування є ключовою складовою будь-якого ci/cd робочого процесу. Автоматизація тестів дозволяє переконатися, що ваш код відповідає стандартам якості та допомагає виявити помилки на ранніх стадіях розробки. Jenkins підтримує широкий спектр фреймворків тестування, і існує багато плагінів для їх інтеграції у ваш pipeline.

н) Централізоване логування. Централізоване логування є важливим для моніторингу та вирішення проблем у масштабних Jenkins інсталяціях. Використання централізованого рішення для логування дозволяє легко

агрегувати та аналізувати логи з кількох екземплярів Jenkins, що допомагає швидше та ефективніше виявляти проблеми.

о) План резервного копіювання та відновлення. Як і будь-яка критична система, Jenkins має мати план резервного копіювання та відновлення. Регулярне резервне копіювання вашого екземпляра Jenkins забезпечує безпеку ваших даних у разі непередбачених подій. Також критично важливо регулярно перевіряти вашу стратегію відновлення, щоб переконатися, що вона працює за планом.

4.4 Інтеграція з іншими інструментами CI/CD

Ефективність тестового конвеєра в значній мірі залежить від його здатності інтегруватися з іншими інструментами та сервісами, що використовуються в процесі розробки та доставки програмного забезпечення. У цьому розділі ми аналізуємо, як розроблений конвеєр на базі Jenkins може бути інтегрований з іншими популярними системами CI/CD, та які особливості цієї інтеграції слід врахувати.

Аналіз сумісності:

– Jenkins і Git – Jenkins тісно інтегрується з Git, який є однією з найпопулярніших систем керування версіями. Ця інтеграція дозволяє автоматизувати запуск пайплайнів Jenkins при оновленнях або створенні pull-запитів. Використання Jenkinsfile дозволяє визначати кожен крок у процесі розгортання Jenkins. Це робить передачу завдань між серверами легшою, оскільки все, що потрібно, - це створити нове завдання у Jenkins, пов'язати його з Jenkinsfile, який зберігається у системі керування версіями, і вказати, коли завдання повинно виконуватися.

– Jenkins і Docker – Для проектів, які використовують контейнеризацію, Jenkins може інтегруватися з Docker для автоматизації збирання образів контейнерів та їх розгортання. Це дозволяє створювати ізольовані та консистентні середовища для кожної фази CI/CD.

– Jenkins і Kubernetes – У випадку використання оркестрації контейнерів через Kubernetes, Jenkins може бути налаштований для управління розгортанням додатків в кластерах Kubernetes, використовуючи плагіни та Jenkinsfile.

– Jenkins і Cloud Services – Jenkins взаємодіє з багатьма хмарними платформами, такими як AWS, Azure та GCP, через плагіни, які дозволяють управляти ресурсами хмари прямо з пайплайнів Jenkins.

– Інтеграція через API та скрипти – Jenkins підтримує розширену інтеграцію через його API, яке дозволяє іншим системам спілкуватися з Jenkins для виконання завдань, отримання статусу збірок та інше. Скрипти на Groovy в Jenkinsfile надають додаткову гнучкість та контроль над пайплайном.

Додатково слід відмітити ключові аспекти інтеграції автоматичних тестів і Jenkins.

– Налаштування ssh ключів. Для забезпечення безперервної інтеграції між репозиторієм (наприклад, у GitLab) і Jenkins, необхідно правильно налаштувати SSH ключі. Це дозволяє безпечну взаємодію між системами і є основою для автоматичної інтеграції коду.

– Створення та конфігурація Jenkins проекту. На наступному етапі створюється новий проект у Jenkins (зазвичай, як "Freestyle project"), де налаштовується джерело коду, вказується SSH URL до репозиторія, а також налаштовуються необхідні облікові дані. Така конфігурація дозволяє Jenkins автоматично реагувати на зміни в репозиторії і запускати процеси збірки та тестування.

– Інтеграція з VCS. Ця частина включає налаштування взаємодії між Jenkins і VCS за допомогою веб-хуків (за наявності). Таке налаштування забезпечує, що Jenkins автоматично повідомляється про нові зміни в репозиторії, що дозволяє відповідно реагувати та запускати процеси тестування.

– Автоматизація тестування. Jenkins може бути налаштований так, щоб автоматично запускати різні типи тестів, включаючи модульні тести, інтеграційні тести та інші, відразу після збірки. Це забезпечує швидке виявлення та вирішення помилок у коді, зменшуючи ризики, пов'язані з розгортанням неякісного програмного забезпечення.

– Управління збірками. Jenkins дозволяє детально налаштовувати процеси збірки, включаючи вибір гілок для збірки, параметри запуску та критерії успішності збірки. Такий підхід забезпечує гнучкість і контроль над процесом розробки і тестування.

4.5 Оцінка продуктивності Jenkins

Jenkins ефективно справляється з різними навантаженнями, і це може бути підтверджено за допомогою моніторингу різних ключових показників продуктивності (KPI). Однією з ключових особливостей Jenkins є його розподілена архітектура Master-Slave, що дозволяє розподіляти навантаження та покращувати загальну продуктивність системи. Master-сервер керує розподілом завдань і станом Slave-вузлів, тоді як Slave-вузли виконують завдання та повертають результати Master-серверу [35].

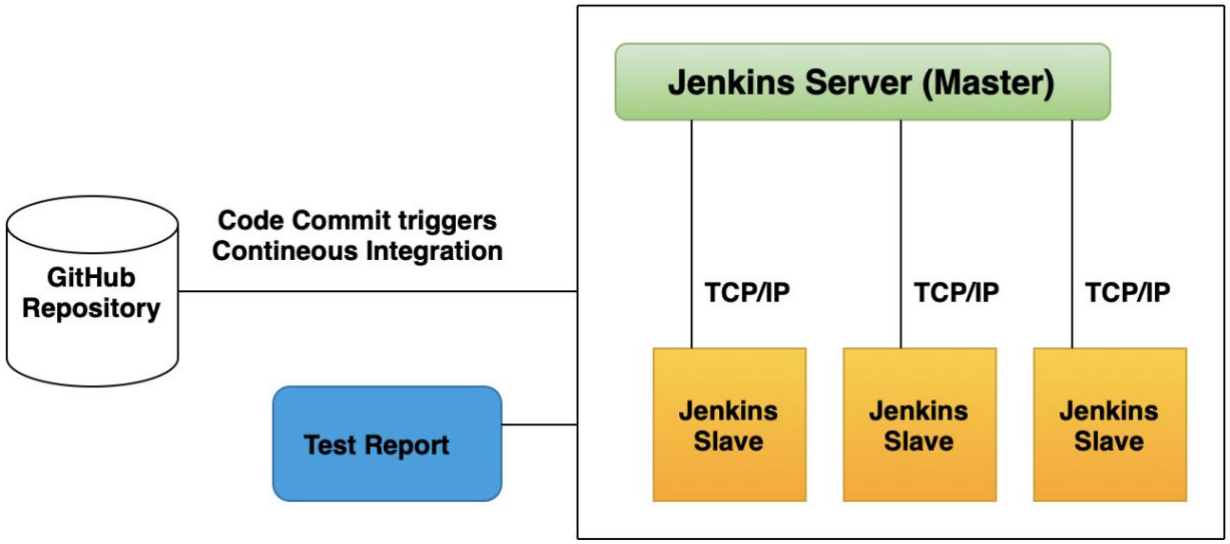

Рисунок 4-2 – Розподільна архітектура Jenkins

Jenkins дозволяє налаштувати різні параметри для оптимізації продуктивності [36]. Наприклад, можна змінювати кількість потоків опитувань чи кількість виконавців, щоб вплинути на загальну продуктивність та пропускну здатність. Моніторинг впливу різних наборів конфігурацій на продуктивність Jenkins допомагає ідентифікувати оптимальний набір параметрів.

Аспекти та методи оптимізації продуктивності Jenkins:

а) Використання останніх версій і резервне копіювання. Рекомендується використовувати останні довготривалі версії підтримки (LTS) Jenkins і плагінів для забезпечення найкращої продуктивності. Також важливо створювати резервні копії конфігураційних файлів Jenkins.

б) Мінімізація кількості збірок на Master Node, оскільки він є "мозком" системи Jenkins, що виконуються на ньому, залишаючи достатньо ресурсів для планування та запуску збірок на вузлах-агентах.

в) Оптимізація історії збірок полягає у обмеженні кількості збережених збірок та тривалості їх зберігання в системі може покращити продуктивність, зменшуючи навантаження на файлову систему.

г) Розумний вибір плагінів має великий вплив на продуктивність Jenkins. Важливо вибирати лише необхідні плагіни та уникати навантаження системи надмірною кількістю плагінів.

д) Інтеграція з іншими інструментами, такими як Gitlab, Github чи Gerrit, може покращити продуктивність, дозволяючи автоматично перевіряти запити та виявляти помилки.

е) Настройка Heap Size для Java-додатків, включаючи Jenkins, є ключовим для покращення продуктивності та запобігання помилкам через нестачу пам'яті.

ж) Оптимізація збору сміття в JVM може покращити продуктивність, зменшуючи затримки, пов'язані з управлінням пам'яттю, особливо в системах з великими об'ємами пам'яті.

з) Використання декількох Master Node для різних проектів або команд може допомогти уникнути конфліктів ресурсів та забезпечити їх специфічне розподілення між командами.

и) Розділення великих робіт на декілька менших може допомогти уникнути помилок та полегшити їх перезапуск у разі необхідності. Це може бути здійснено за допомогою плагіна Workflow.

к) Оптимізація Зберігання: Забезпечення достатнього дискового простору є критичним для ефективного функціонування Jenkins, особливо при зростанні використання системи.

л) Видалення старих даних та управління ними може покращити продуктивність, уникнути проблем з пам'яттю та прискорити реакцію інтерфейсу користувача.

м) Участь у спільноті Jenkins та використання доступних ресурсів та порад може допомогти підтримувати оптимальну роботу Jenkins та слідувати кращим практикам.

Для моніторингу продуктивності Jenkins можна використовувати плагін JavaMelody, який дозволяє відслідковувати різні показники, такі як використання CPU, використання пам'яті та час очікування в черзі збирання

[37]. Цей плагін надає візуалізації цих метрик у реальному часі, дозволяючи швидко виявляти тренди та потенційні проблеми. Крім того, JavaMelody дозволяє налаштувати сповіщення за певними пороговими значеннями критичних метрик, таких як використання CPU або пам'яті, що дозволяє швидко реагувати на проблеми перед тим, як вони вийдуть з-під контролю.

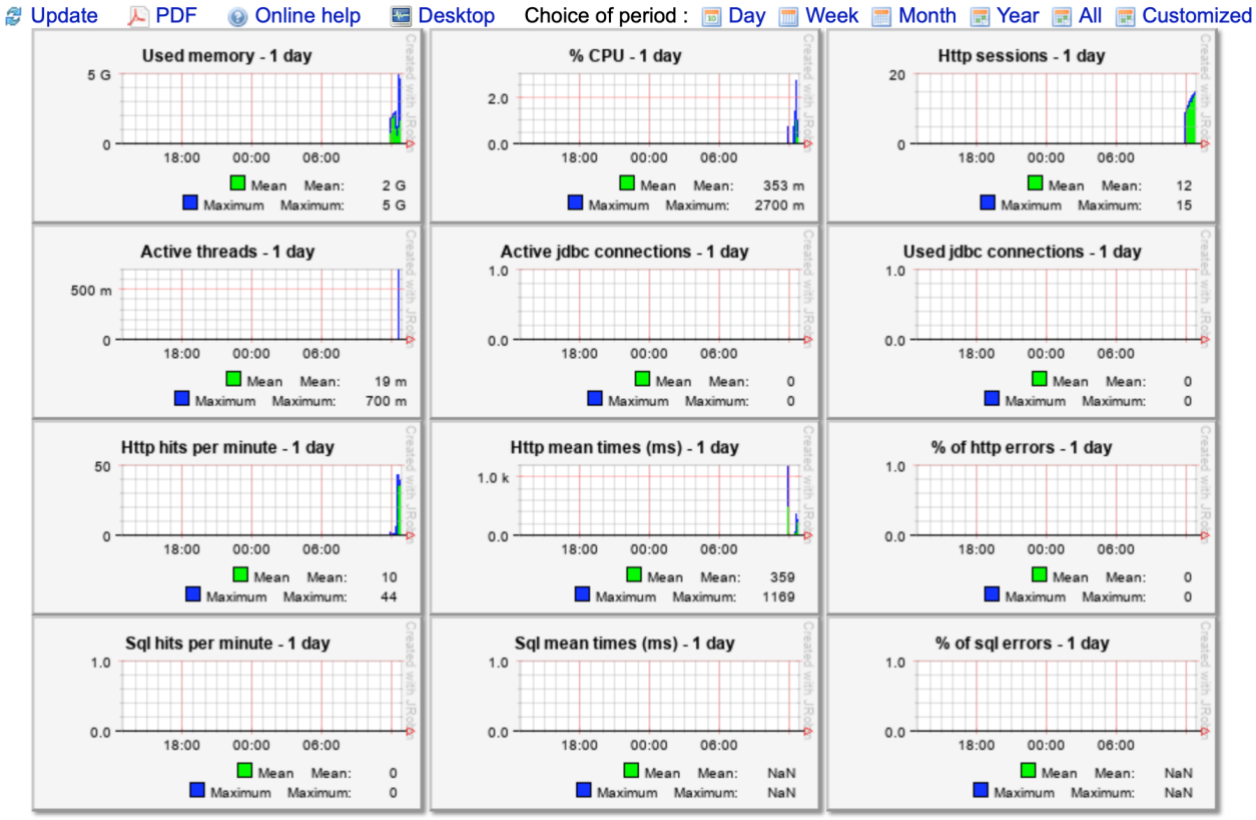

Рисунок 4-3–Візуалізація роботи плагіну JavaMelody

JavaMelody допомагає ідентифікувати та вирішувати загальні проблеми продуктивності Jenkins, такі як:

– Повільні збірки
– Часті збої
– Проблеми з плагінами

За допомогою даних, наданих JavaMelody, можна аналізувати тренди та порівнювати їх з очікуваними показниками для виявлення основних причин проблем.

Для оптимального використання JavaMelody, важливо дотримуватися кращих практик:

– Робити фокус на основних метриках, таких як використання CPU та пам'яті.

– Встановлювати чіткі пороги для сповіщень.

– Регулярно перевіряти метрики Jenkins.

– Використовувати історичні дані для виявлення трендів і вдосконалення стратегії моніторингу.

4.6 Тенденції майбутнього розвитку конвеєрів

У контексті майбутнього розвитку тестових конвеєрів, особливо враховуючи Jenkins, декілька ключових тенденцій впливатимуть на їхню еволюцію [38,39]:

а) Інтеграція ШІ та машинного навчання. Очікується, що штучний інтелект і машинне навчання будуть інтегровані в CI/CD для автоматизації та поліпшення процесів. Це може включати прогнозування та запобігання інтеграційним проблемам та оптимізацію процесів тестування та розгортання.

б) Перехід до мікросервісів та безсерверних архітектур. Мікросервіси та безсерверні архітектури набирають популярності, пропонуючи більшу масштабованість, гнучкість та стійкість.

в) Зростання платформ CI/CD, що самообслуговуються. Самообслуговуванні платформи спрощують процес CI/CD, роблячи його доступнішим для розробників.

г) Підтримка розробки для різних платформ. Зростає потреба в розробці програмного забезпечення, яке працює на різних платформах та пристроях.

д) Використання низькокодових та безкодових інструментів у CI/CD. Ці інструменти можуть допомогти командам спростити процеси та зменшити ризик помилок.

е) Зростання Edge Computing у розгортанні. Це тенденція включає переміщення обробки даних та зберігання ближче до джерел даних, що прискорює час відгуку і зменшує використання смуги пропускання. У контексті CI/CD це стає все більш важливим для розробників, щоб надійно доставляти останні версії програм та сервісів на ці пристрої.

ж) Розвиток тестування з допомогою ШІ у CI/CD. [39] Алгоритми машинного навчання можуть автоматично генерувати велику кількість

тестових випадків, охоплюючи різні шляхи та входи в програмному забезпеченні, що забезпечує більш повне покриття тестами. ШІ може прискорити процес тестування, дозволяючи швидше віддавати зворотний зв'язок розробникам, що зменшує загальний час тестування. ШІ може автоматизувати такі завдання, як створення тестових випадків та підготовка даних, що дозволяє тестувальникам зосередитись на більш складних завданнях. ШІ може використовувати прогностичну аналітику для виявлення ймовірних помилок у коді та забезпечувати реальний моніторинг програмного забезпечення, виявляючи незвичайну поведінку або помилки продуктивності.

з) Покращена автоматизація у керуванні інфраструктурою.З розвитком хмарних технологій та архітектур, таких як Kubernetes та серверлес, управління інфраструктурою стає складнішим. Ефективні інструменти автоматизації будуть вирішальними для управління цією складністю, дозволяючи командам зосередитися на доставці цінності клієнтам.

и) Автоматизований перегляд коду. Автоматизований перегляд коду стане ще поширенішим у 2024 році, оскільки він сприяє швидкості та якості доставки програмного забезпечення. Автоматизовані перевірки коду допомагають виявити проблеми на ранніх етапах розробки, зменшуючи витрати та зусилля на їх виправлення.

к) Серверлес CI/CD конвеєри. Серверлес технології дозволяють розробникам зосередитися на своєму коді, не переймаючись про базову інфраструктуру. Це призводить до швидшої доставки та зниження витрат.

л) Посилення інтеграції безпеки в CI/CD. Безпека є основною турботою у сфері розробки програмного забезпечення. До 2024 року можна очікувати на більш тісну інтеграцію елементів безпеки в ci/cd конвеєри.

м)Спостереження у режимі реального часу та швидкі відгуки. Моніторинг у реальному часі та миттєві зворотні зв'язки стануть ще більш критичними в CI/CD. Це дозволить швидше виявляти та виправляти проблеми, забезпечуючи безперебійну роботу конвеєру.

н) Розвиток систем самовідновлення. Системи самовідновлення, які виявляють проблеми та виправляють їх автоматично, можуть кардинально змінити ci/cd, зменшуючи потребу в ручному моніторингу та усуненні неполадок.

ВИСНОВКИ

У цій дипломній роботі було здійснено комплексне дослідження впровадження автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення, з особливим фокусом на Jenkins та інші інструменти CI/CD. Аналіз підкреслив, що інтеграція та автоматизація є критичними елементами для підвищення продуктивності та якості в розробці ПЗ. Jenkins, з його великою кількістю можливостей для розширення та високим ступенем гнучкості, виступає як важливий інструмент для підтримки різних аспектів безперервної інтеграції та доставки. Інтеграція CI/CD є ключовим фактором у підвищенні швидкості та якості розробки, дозволяючи здійснювати частіші оновлення та випуски продуктів.

Робота зосереджена на тому, як автоматизовані тестові конвеєри сприяють зменшенню часу на розробку, поліпшенню якості продукту та оптимізації витрат. Важливість Jenkins та інших інструментів CI/CD полягає не лише у їх функціональності, але й у здатності інтегруватися з різноманітними системами та службами, що робить їх незамінними для сучасних процесів розробки. Jenkins та інші інструменти CI/CD демонструють значну гнучкість та масштабованість, що є критично важливими для підтримки швидкого зростання та розширення проектів. Здатність адаптуватися до змінних вимог проекту та робочого навантаження є ключем до успішного випуску продукту.

У дослідженні також висвітлені майбутні тренди в інструментах CI/CD та їх потенційний вплив на методи розробки. Окрім того, робота включає відгуки користувачів Jenkins, надаючи практичну перспективу його застосування та переваг. Ці відгуки відображають загальну позитивну оцінку Jenkins в індустрії.

Використання автоматизованих тестових конвеєрів сприяє культурі неперервного навчання та співпраці в командах розробників. Завдяки цьому підвищується рівень обміну знаннями та взаємодії між різними відділами компанії, що позитивно впливає на загальну продуктивність та інноваційність.

Продовжуючи інтегрувати інноваційні технології, такі як штучний інтелект та машинне навчання, інструменти CI/CD, включаючи Jenkins, можуть підвищити ефективність та точність процесів розробки. Це також сприяє покращенню якості програмного продукту та скороченню часу на впровадження нових функцій. ШІ обіцяє значно оптимізувати процеси

тестування, прогнозувати та виправляти помилки, а також адаптувати тестування під конкретні потреби проектів.

На основі аналізу можна зробити висновок, що Jenkins та подібні інструменти продовжать грати ключову роль у розвитку та оптимізації процесів безперервної інтеграції та доставки, адаптуючись до змінних потреб розробників та технологічних інновацій. У цьому контексті, важливим є постійне вивчення нових підходів, технологій, та інструментів у сфері CI/CD, щоб підтримувати інновації та ефективність в розробці програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ольга Чорна, Юрій Нетребін «Огляд перспектив застосування автоматизованих тестових конвеєрів у безперервній розробці програмного забезпечення» //12-та Міжнародна науково-технічна конференція «ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ ІСТ-2023» – 2023. – С. 69-73.
2. Farley, Jez Humble and David. Continuous delivery : reliable software releases through build, test, and deployment automation. 501 Boylston Street, Suite 900, Boston, MA 02116 : Pearson Education, Inc, 2011. ISBN 978-0-321-60191-9.
3. Amazon Web Services, Inc., or its affiliates. What is DevOps? AWS Site. [Електронний ресурс] – https://aws.amazon.com/devops/.– Дата доступу 8.11.2023
4. What is Continuous Integration? AWS Site. [Електронний ресурс] https://aws.amazon.com/devops/continuous-integration/.– Дата доступу 8.11.2023
5. What is Continuous Delivery? AWS Site. [Електронний ресурс] https://aws.amazon.com/devops/continuous-delivery/.– Дата доступу 8.11.2023
6. IBM Cloud Education. What Are CI/CD and the CI/CD Pipeline? [Електронний ресурс] – https://www.ibm.com/blog/ci-cd-pipeline/.– Дата доступу 8.11.2023
7. JetBrains s.r.o. [Електронний ресурс] https://www.jetbrains.com/lp/devecosystem-2022/team-tools/.– – Дата доступу 18.10.2023
8. JetBrains s.r.o. Testing - The State of Developer Ecosystem in 2022 Infographic | JetBrains: Developer Tools for Professionals and Teams. JetBrains. [Електронний ресурс] https://www.jetbrains.com/lp/devecosystem-2022/testing/.– Дата доступу 9.10.2023
9. Технічний комітет стандартизації «Інформаційні технології» (ТК 20). ДСТУ ISO/IEC/IEEE 29119-1:2017 Інженерія систем і програмних засобів. Тестування програмних засобів. Частина 1. Поняття та визначення (ISO/IEC/IEEE 29119-1:2013, IDT).

10. Веб-системи в освіті: основні методи та принципи проведення тестування юзабіліті. СВ Тітов, ЕВ Тітова. 2015 р. ББК 74.580. 22в. 62я431 Э41.
11. Whittaker, J. SwTestng.pdf. “Software Testing: What it is and Why it is So Difficult”. [Електронний ресурс] – www.se.fit.edu/papers/SwTestng.pdf. – Дата доступу 18.10.2023
12. Dustin, Elfriede та al, et. Implementing Automated Software Testing. Addison Wesley, 2009. ISBN 978-0-321-58051-1.
13. Docker Inc. docker. docker. [Електронний ресурс] https://www.docker.com. – Дата доступу 1.11.2023
14. Foundation, ehe OWASP. OWASP Guide Project. www.owasp.org. [Електронний ресурс] – http://www.owasp.org/index.php/OWASP_Guide_Project– Дата доступу 10.11.2023.
15. Hamilton, Thomas. Types of Software Testing (100 Examples). www.guru99.com. [Електронний ресурс] – https://www.guru99.com/types-of-software-testing.html. – Дата доступу 10.11.2023
16. Pan, Jiantao. Software Testing. Carnegie Mellon University. [Електронний ресурс] – http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/.– Дата доступу 10.11.2023
17. Singh, Rohan. The Testing Pyramid: Simplified for One and All. www.headspin.io. [Електронний ресурс] – https://www.headspin.io/blog/the-testing-pyramid-simplified-for-one-and-all. – Дата доступу 10.11.2023
18. GitHub, Inc. GitHub Actions documentation - GitHub Docs. [Електронний ресурс] – https://docs.github.com/ru/actions. – Дата доступу 10.11.2023
19. Dancuk, Milica. What is Jenkins? [Електронний ресурс] https://phoenixnap.com/kb/what-is-jenkins. – Дата доступу 1.11.2023
20. GitLab B.V. Continuous Integration and Delivery. [Електронний ресурс] – https://about.gitlab.com/solutions/continuous-integration/.– Дата доступу 10.11.2023
21. Travis CI. Travis CI. [Електронний ресурс] – https://www.travis-ci.com. – Дата доступу 1.11.2023
22. Atlassian. Bitbucket Pipelines - Continuous Delivery | Bitbucket. [Електронний ресурс] – https://bitbucket.org/product/features/pipelines. – Дата доступу 11.11.2023

23. Amazon Web Services, Inc. . Developer tools. [Електронний ресурс] – https://aws.amazon.com/ru/products/developer-tools/. – Дата доступу 11.11.2023
24. JetBrains s.r.o. TeamCity. [Електронний ресурс] – https://www.jetbrains.com/teamcity/. – Дата доступу 1.11.2023
25. Cloud Build. [Електронний ресурс] – https://cloud.google.com/build. – Дата доступу 1.11.2023
26. JetBrains s.r.o. JetBrains Space: The Intelligent Code Collaboration Platform. [Електронний ресурс] – https://www.jetbrains.com/space/.– Дата доступу 11.11.2023
27. Harness Inc. Drone CI – Automate Software Testing and Delivery. [Електронний ресурс] – https://www.drone.io. – Дата доступу 11.11.2023
28. Appveyor Systems Inc. Continuous Integration and Deployment service for Windows, Linux and macOS | AppVeyor. [Електронний ресурс] – https://www.appveyor.com. – Дата доступу 1.11.2023
29. Thoughtworks Inc. under the Apache License, Version 2.0. Open Source Continuous Delivery and Release Automation Server | GoCD. [Електронний ресурс] – https://www.gocd.org/.– Дата доступу 12.11.2023
30. Douglas, Brian. 4 ways we use GitHub Actions to build GitHub. [Електронний ресурс] – https://github.blog/2022-04-05-4-ways-we-use-github-actions-to-build-github/. – Дата доступу 12.11.2023
31. Tong, Alyssa. 3 Cases: Jenkins success stories from the community. [Електронний ресурс] – https://www.jenkins.io/blog/2020/12/05/3-Cases-Jenkins-Success-stories-from-the-community/.– Дата доступу 12.11.2023
32. GitLab B.V. Customer stories. [Електронний ресурс] – https://about.gitlab.com/customers/.– Дата доступу 12.11.2023
33. GUVI Geeks Network Pvt. Ltd. Real-World Automation Testing Applications. [Електронний ресурс] https://www.guvi.in/blog/real-world-automation-testing-applications/.– Дата доступу 12.11.2023
34. Christiano, Paul. Automation Testing Tools In 2023: An In-Depth Guide To Code-Based, Robotics & AI Tools. [Електронний ресурс] https://expertbeacon.com/automation-testing-tools/.– Дата доступу 12.11.2023
35. Pipeline. [Електронний ресурс] – https://www.jenkins.io/doc/book/pipeline/.– Дата доступу 12.11.2023

36. baeldung. Guide to Jenkins Architecture and Performance Improvements. [Електронний ресурс] – https://www.baeldung.com/ops/jenkins-performance. – Дата доступу 12.11.2023
37. Orbach, Tidhar Klein. 5 Simple tips for boosting your Jenkins performance . [Електронний ресурс] – https://blog.taboola.com/5-simple-tips-boosting-jenkins-performance/.– Дата доступу 12.11.2023
38. Site24x7. Jenkins monitoring guide. Site24x7. [Електронний ресурс] – https://www.site24x7.com/learn/jenkins-performance-monitoring.html. – Дата доступу 1.11.2023
39. CI/CD Pipeline Trends for 2023: Continuous Integration & Delivery Strategy. [Електронний ресурс] – https://katalon.com/resources-center/blog/ci-cd-pipeline-trends. – Дата доступу 1.11.2023
40. Maayan, Gilad David. CI/CD: Trends and Predictions for 2024. [Електронний ресурс] – https://www.techdee.com/ci-cd-trends-for-2024/.– Дата доступу 12.11.2023
41. Zinchenko, Vladyslav. Constant Readiness or What is CI/CD in DevOps. [Електронний ресурс] – https://dataforest.ai/blog/constant-readiness-or-what-is-ci-cd-in-devops. – Дата доступу 15.11.2023