Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Система безперервного розгортання проектів для оптимізації роботи компанії зі створення комерційного програмного забезпечення
(тема)

Виконав:

студент II курсу, групи СПм-22-2

Шепета О. В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Федорченко В. М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

# ЗАВДАННЯ

## НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Шепеті Олександру Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система безперервного розгортання проектів для оптимізації роботи компанії зі створення комерційного програмного забезпечення

затверджена наказом по університету від "06" листопаду 2023 р. № 1299Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи Нормативно-правові та законодавчі акти України, фахові періодичні видання з комп'ютерних наук (інформаційні системи та технології), науково-методичні розробки та результати, що оприлюднено в працях вітчизняних та зарубіжних авторів. Використовувати середовище об'єктно-орієнтованого проектування Visual Studio 2022, мову програмування C#, C++, Java, PHP.

4. Перелік питань, що потрібно опрацювати у роботі

1) теоретичні основи безперервного розгортання проектів у процесі розробки;
2) аналіз та оцінка показників конкурентної спроможності підприємств;
3) побудова моделі оцінки впливу впровадження системи безперервного розгортання на на показники конкурентоспроможності.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) загальний вигляд взаємодії модулів систем безперервної інтеграції у процесі розробки програмного забезпечення, алгоритм роботи системи безперервного розгортання, співвідношення популярності систем безперервного розгортання на ринку IT, слайд-презентація – 13 слайдів

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

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Розробка плану кваліфікаційної роботи, | 7.11.23-13.11.23 | |
| | з літературними джерелами за темою | | |
| 2 | Аналіз існуючих методів вирішення поставлених | 14.11.23-20.11.23 | |
| | завдань, написання теоретичної частини | | |
| | кваліфікаційної роботи | | |
| 3 | Розроблення методу (моделі) вирішення завдання з | 21.11.23-23.11.23 | |
| | тестовим прикладом його застосування в ручний | | |
| | спосіб | | |
| 4 | Розроблення та тестування програмного продукту | 24.11.23-06.12.23 | |
| | реалізації методу (моделі) | | |
| 5 | Проведення експериментів та їх аналіз з описом та | 7.12.23-23.12.23 | |
| | обґрунтуванням отриманих результатів | | |
| 6 | Оформлення матеріалів кваліфікаційної роботи | 26.12.23-02.01.24 | |
| 7 | Подання кваліфікаційної роботи керівникові та її | 03.01.24-06.01.24 | |
| | попередній захист | | |
| 8 | Подання кваліфікаційної роботи на рецензування | 09.01.24-12.01.24 | |

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Федорченко В. М.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 92 с., 43 рис., 16 табл., 2 дод., 75 джерел.

СИСТЕМИ БЕЗПЕРЕРВНОГО РОЗГОРТАННЯ ІТ-ПРОЕКТІВ, МЕТОДИ, МОДЕЛІ, КОНКУРЕНТОСПРОМОЖНІСТЬ, ПОКАЗНИКИ КОНКУРЕНТОСПРОМОЖНОСТІ ІТ-КОМПАНІЙ, АВТОМАТИЗАЦІЯ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метою кваліфікаційної роботи є підвищення показників конкурентоспроможності на основі впровадження системи безперервного розгортання проектів та оптимізація роботи компанії зі створення комерційного програмного забезпечення.

Об’єктом дослідження є процес підвищення конкурентоспроможності ІТ компанії на основі збільшення продуктивності команди за рахунок впровадження системи безперервного розгортання проектів.

Предмет дослідження – економічний механізм формування конкуренто-спроможності та вплив впровадження системи безперервного розгортання проектів на показники ефективності.

У ході виконання кваліфікаційної роботи розглянуто особливості автоматизації процесу розробки комерційного програмного забезпечення за допомогою системи безперервного розгортання проектів та вплив впровадження систем автоматизації на показники конкурентоспроможності ІТ-компаній.

Отримані результати можуть бути використані як ІТ-компаніями для розуміння важливості автоматизації розробки програмного забезпечення, так і студентами як вектор розвитку майбутнього бізнесу у сфері інформаційних технологій.

# ABSTRACT

Master's thesis: 92 pages, 43 figures, 16 tables, 2 appendices, 75 sources.

KEYWORDS: A SYSTEM OF CONTINUOUS DEPLOYMENT OF IT-PROJECTS, METHODS, MODELS, COMPETITIVENESS OF, INDEX OF COMPETITIVENESS OF IT-COMPANIES, AUTOMATING THE PROCESS OF SOFTWARE DEVELOPMENT.

The major goal of this thesis is to increase competitiveness indicators through the introduction of a system of continuous deployment of projects and optimization of the company to build commercial software.

The object of research is the process of increasing the competitiveness of IT-companies based on the increase in team productivity through the introduction of continuous deployment projects.

The subject of research – economic mechanism of formation of competitiveness and the impact of the introduction of a system of continuous deployment to the performance indicators of projects.

In work features of activity of automating the process of developing commercial software using a continuous deployment project, as well as the impact of the introduction of automation systems on indicators of competitiveness of IT-companies.

The results can be used as the IT-companies to understand the importance of automating software development, and students as a vector of development of the future of business in the sphere of information technology.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GIT – система контрлю версій

CI – безперервна інтеграція (англ., Continuous Integration)

Backlog – черга завдань

XML – мова розмітки, що розширюється

SQL – мова структурованих запитів (англ., Structured Query Language)

# ВСТУП

У будь якому підприємстві, незалежно від виду та сфери діяльності гостро повстають питання ефективного використання економічного потенціалу, конкурентоспроможності та автоматизації бізнес-процесів. В даний час все менше завдань, які виконуються без використання різноманітних програмних засобів як в державному секторі управління, бізнесі, так і в приватному житті. При цьому інформаційні системи все глибше проникають в усі сфери нашого життя, що веде до прагнення автоматизації максимальної кількості процесів як для спрощення і прискорення обробки інформації, так і для підвищення безпеки даних. Подібна тенденція неодмінно веде до "обтяження" програмних продуктів, що забезпечують виконання поставлених цілей, а також і до ускладнення їх написання.

При розробці інформаційної системи використовують каскадну або водоспадну модель розробки програмного забезпечення, що передбачає послідовне виконання етапів діяльності. В даному випадку, після визначення вимог і проектування до етапу реалізації (кодування) складного програмного продукту приступає кілька груп фахівців. Створені частини системи перед етапом тестування та впровадження повинні бути зібрані в більші модулі і об'єднані в єдину систему, однак, процес складання починається тільки тоді, коли модулі практично повністю готові. Зрозуміло, що на даній стадії в коді виявляються помилки або виникають проблеми взаємодії окремих частин системи, що веде до великих затрат праці на їх усунення. Окрім цього, до самого закінчення етапу розробки і збірки модулів абсолютно не відомий майбутній обсяг можливих помилок, а відповідно і обсяг необхідного часу на їх виправлення, що може привести до збільшення термінів розробки проекту і в цілому вартості системи для замовника. Одним з можливих способів вирішення даної проблеми є використання методології безперервної інтеграції (Continuous Integration) при розробці програмного забезпечення різних сфер використання.

Актуальність роботи полягає в тому, що на сьогоднішній день все більшу значимість набувають процеси автоматизації робіт, які часто

виконуються, а також робіт, що не піддаються формалізації. Розробка процесу автоматизації за рахунок сервера безперервної інтеграції дозволить підвищити ефективність і оперативність діяльності співробітників на різних стадіях обробки інформаційних потоків, а також в значній мірі вплине на конкурентоспроможність IT-компанії в цілому.

Також потрібно звернути увагу на те, що результати дослідження можуть бути використані і самими IT-компаніями, для того щоб оцінити становище підприємства у плані економічного потенціалу, побудові внутрішніх процесів та оцінки автоматизації діяльності загалом.

Мета даної роботи полягає у тому, щоб звернути увагу на вузькі ділянки в процесі розробки, що дозволить компанії мати конкурентну перевагу та виділити свою діяльність серед перенасиченого ринку розробки комерційного програмного забезпечення. Розробка та впровадження системи безперервного розгортання проектів дозволить підвищити показники ефективності та уникнути ряду помилок, що призводять до зниження економічного потенціалу.

Об’єкт дослідження – діяльність IT компанії, а саме процес розробки та підвищення продуктивності за рахунок впровадження систем автоматизації розгортання коду. Це дозволяє підвищити показники конкурентоспроможності та вивести діяльність підприємства на новий рівень роботи з клієнтами. В даному випадку замовник повністю залучений до процесу розробки.

Предмет дослідження – економічний механізм формування конкуренто-спроможності та вплив впровадження системи безперервного розгортання проектів на показники ефективності.

# 1 ТЕОРЕТИЧНІ ОСНОВИ БЕЗПЕРЕРВНОГО РОЗГОРТАННЯ ПРОЕКТІВ У ПРОЦЕСІ РОЗРОБКИ

## 1.1 Поняття, сутність та види систем безперервного розгортання

Щоб випустити програмний продукт, який готовий до використання, замало просто написати код. Після того, як програмісти завершили свою роботу, потрібно ще досить багато часу, щоб представити продукт широкому колу користувачів. Нерідко програмісти навіть не уявляють собі, скільки часу займають рутинні операції, такі як: об'єднати усі модулі майбутнього продукту, що написані різними розробниками, створити інсталятор, провести автоматичне тестування, підготувати документацію. Часто виникає така ситуація: всі поспішають, і тим самим тільки множать кількість помилок і недоліків, на усунення яких теж потрібен певний час – і реліз версії продукту доводиться відкласти на невизначений термін.

Раніше схема розробки будь якого проекту, до впровадження системи безперервної інтеграції, включала наступний порядок: розробник писав новий код, зберігав у системі контролю версій, запускав скрипт для побудови та тестування. Якщо виявлялася яка-небудь помилка, він починав її виправлення [4, 7]. Орієнтовна схема представлена на рисунку (рисунок 1.1).

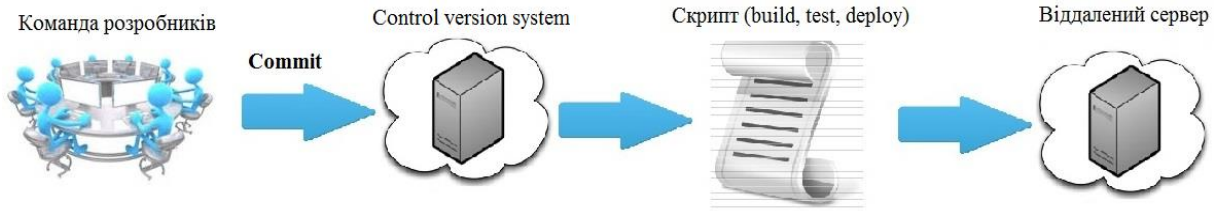


Рисунок 1.1 – Схема розробки програмного забезпечення без використання безперервної інтеграції

Git – це розподілена система контролю версій. На відміну від централізованих і локальних систем, Git не просто вивантажує останні версії файлів, але і повністю копіює репозиторій локально. У разі, коли виникає

збій головного серверу, через який йшла робота, будь-який клієнтський репозиторій може бути скопійований назад на сервер, щоб відновити базу даних та роботу проекту. Завдяки хешуванню, Git легко відстежує зміни в коді проекту.

Ситуація, коли хтось непомітно вносить зміни в код, неможлива. Подібний контроль дозволяє в будь-який момент повернутися до потрібної версії збірки. В результаті, можливо точно дізнатися, з яких початкових кодів збирається проект, що в свою чергу виключає будь-який момент випадковості [53].

Одна з функцій системи контролю версій полягає в можливості створювати кілька гілок (branches) для підтримки декількох напрямків розробки проекту. Рекомендується створювати якомога менше додаткових гілок, зокрема, бажано завести "головну" гілку, як єдину гілку для розробки, а інші створювати для виправлення помилок або тимчасових експериментів. Після написання коду "вторинна гілка", попередньо пройшовши перевірку, додається в загальне сховище і вливається в основний потік розробки. На рисунку (рисунок 1.2) наведений приклад гілок проекту в процесі розробки.

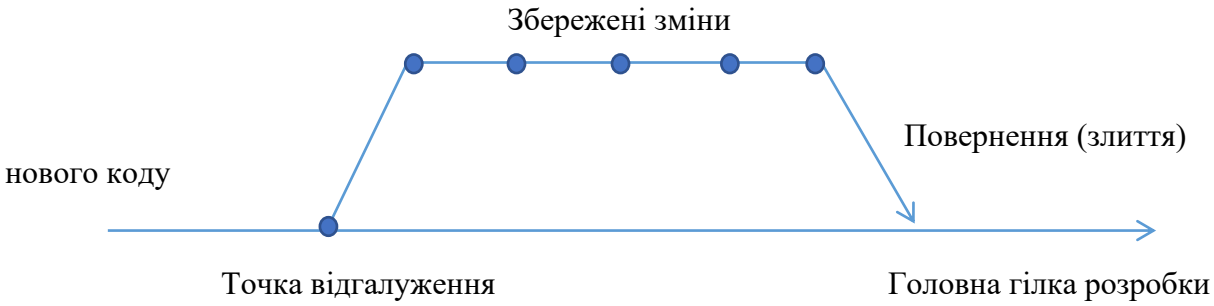


Рисунок 1.2 – Приклад гілок проекту в процесі розробки у системі контролю версій

Безперервна інтеграція – це практика розробки комерційного програмного забезпечення, що полягає у автоматизованому інтегруванні програмних модулів в єдину систему з використанням програмних тестів для уникнення помилок [6, 23].

При розробці проекту, де задіяно декілька різних спеціалістів і вони працюють незалежно, стадія інтеграції є завершальною. Впровадження

системи безперервного розгортання проектів дозволяє знизити трудомісткість інтеграції і зробити її передбачуваною за рахунок раннього виявлення та усунення помилок і суперечностей. Іншими словами безперервна інтеграція – це практика (теорія) розробки програмного забезпечення, при якій учасники команди розробників часто об'єднують результати своєї роботи, від одного до декількох разів на добу. Кожна інтеграція перевіряється автоматичним складанням і тестуванням для виявлення помилок, які могли з'явитися в новій збірці. Подібний підхід істотно зменшує кількість проблем при інтеграції та дозволяє розробляти програмні продукти більш швидко і якісно. Етапи безперервної інтеграції представлені на рисунку (рисунок 1.3).


Рисунок 1.3 – Етапи безперервної інтеграції

Обов’язкові вимоги до проекту з автоматизованим розгортанням:

1) код проекту та все що необхідно для побудови та тестування проекту, зберігається в репозиторії системи контролю версій;

2) операції копіювання з репозиторію, складання і тестування всього проекту автоматизовані і викликаються із зовнішньої програми.

Системи безперервного розгортання проектів використовуються такими гігантами у сфері комерційного програмного забезпечення як Yandex, Google, Instagram, Facebook, адже дані системи мають значні переваги:

1) безперервне розгортання дозволяє швидше працювати програмістам над проектом, так як вони не обмежені декількома розгорнутими версіями в фіксований час, що дозволяє їм вносити зміни по мірі необхідності;

2) дана система дозволяє легше виявляти помилки, які потрапили в той чи інший коміт. Програмістам не доводиться разом аналізувати сотні змін, для того, щоб знайти причину нової помилки, достатньо проаналізувати декілька останніх комітів;

3) коміти, які містять помилки можна швидко виправляти, так як усі сирцеві коди зберігаються у системі контролю версій, з якою і працює система безперервного розгортання [12, 18].

У практиці безперервної інтеграції можна виділити кілька основних етапів розробки IT-рішення: технічне завдання, аналіз, розробка, тестування та випуск (експлуатація). На рисунку 1.4 представлені етапи розробки програмного забезпечення.


Рисунок 1.4 – Етапи розробки програмного забезпечення

У свою чергу взаємодія на рівні блоків розробки та тестування можна розділити на наступні під етапи: створення і зміна частин коду, їх збірка (компоновка, компіляція або інтерпретування), тестування та оформлення робочої копії. Цикл розробки представлений на рисунку 1.5.


Рисунок 1.5 – Цикл розробки програмного забезпечення

З впровадженням системи безперервної інтеграції, процес розробки буде складатися з наступних етапів, які наведено на рисунку (рисунок 1.6).

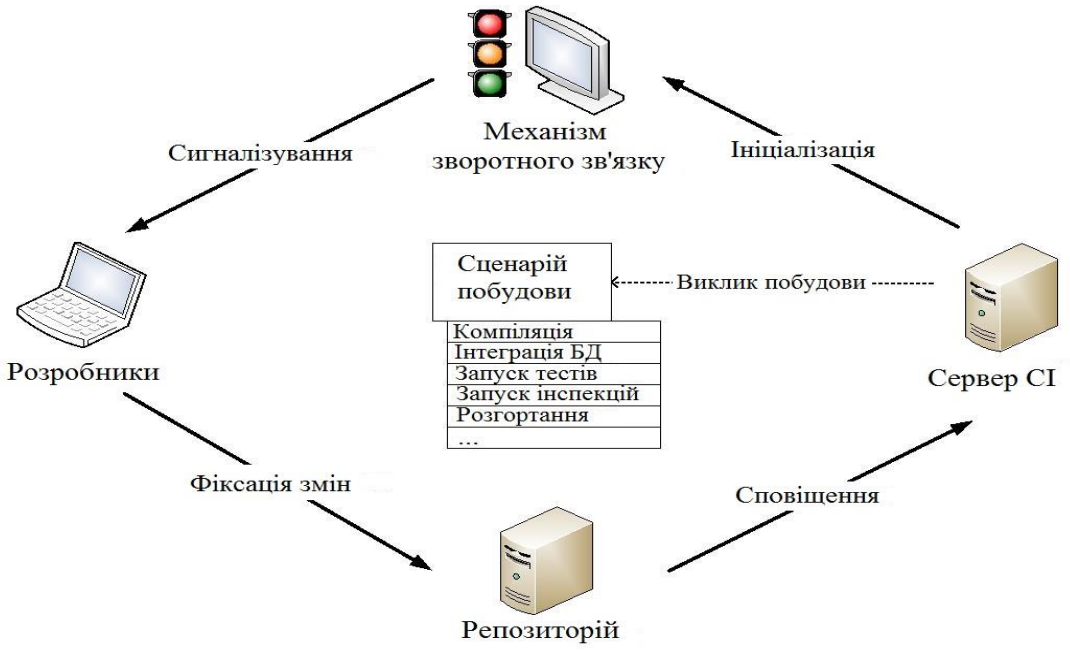


Рисунок 1.6 – Етапи процесу розробки з впровадженням безперервного розгортання

Загальний алгоритм роботи системи безперервного розгортання проектів представлений на рисунку 1.7.

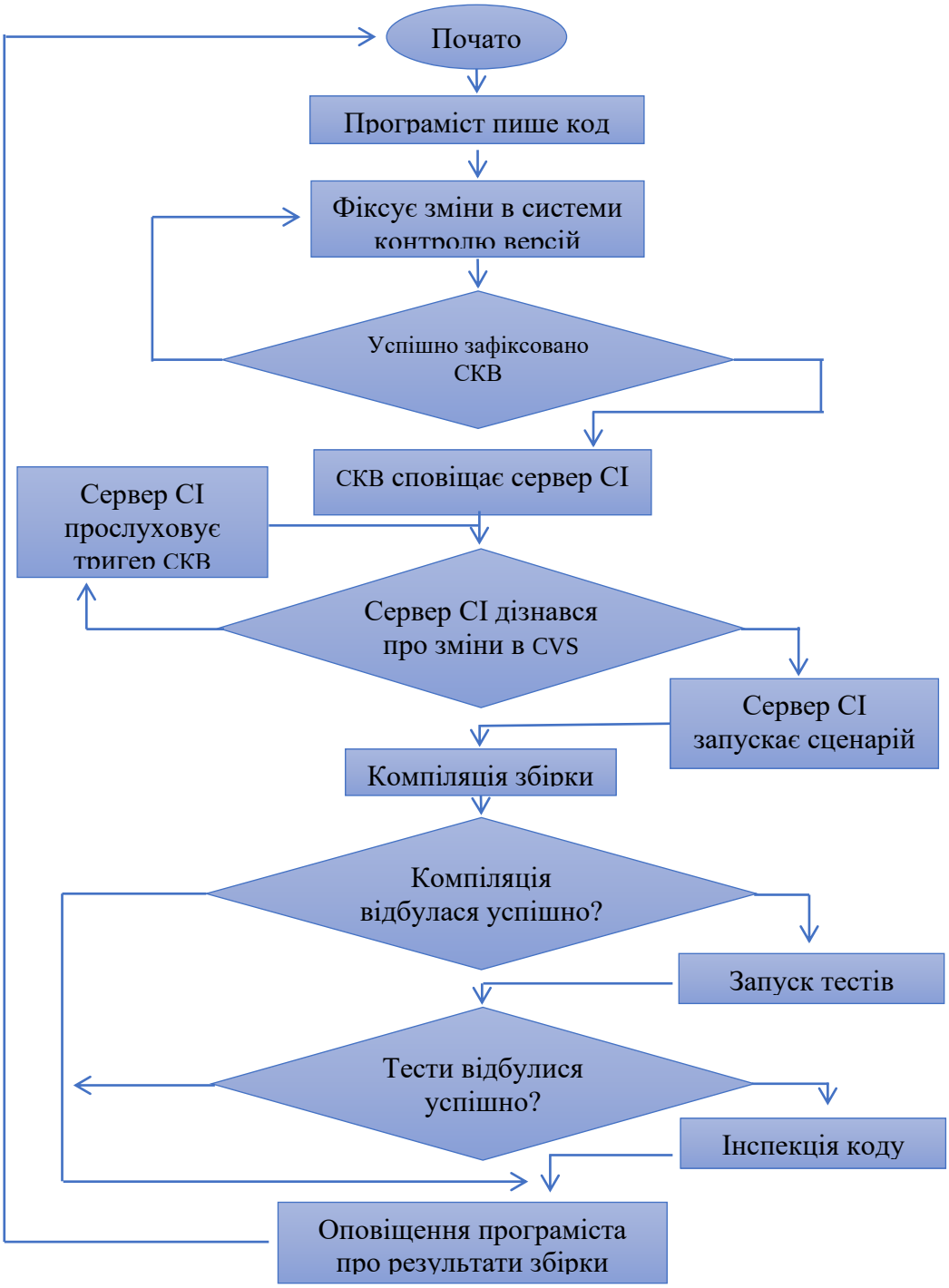


Рисунок 1.7 – Алгоритм роботи системи безперервного розгортання проектів

На виділеному інтеграційному сервері організовується служба, до завдань якої належать: отримання вихідного коду зі сховища; збірка проекту;

виконання тестів; розгортання готового проекту; формування та відправка звітів [28, 29].

Робота з репозиторієм системи контролю версій включає наступні пункти:

1) всі дані зберігаються в репозиторії;

2) часті збереження кодової бази;

3) локальна збірка перед збереженням кодової бази;

4) збірка на інтегрованому сервері;

5) припинення роботи з репозиторієм до виправлення помилок.

Збірка проекту може здійснюватися за наступними етапами:

1) за зовнішнім запитом, тобто програміст може вручну запустити збірку проекту;

2) за розкладом – збірка проводяться щоночі в автоматичному режимі –"нічні збірки" (для того, щоб до початку робочого дня були готові результати тестування і робоча версія проекту). Для відмінності додатково вводиться система нумерації збірок – зазвичай, кожна збірка нумерується натуральним числом, яке збільшується з кожною новою збіркою. Вихідні тексти та інші вихідні дані при взятті їх з репозиторія (сховища) системи контролю версій позначаються номером збірки. Завдяки цьому, точно така ж збірка може бути відтворена в майбутньому – необхідно лише взяти вихідні дані з потрібної позначки і запустити процес знову. Це дає можливість повторно видавати навіть дуже старі версії програми з невеликими виправленнями;

3) за фактом поновлення сховища. Збірка автоматично запускається при успішному збереження у гілку в системі контролю версій;

4) за іншими критеріями, вказаними в налаштування системи безперервного розгортання.

Для безперервної інтеграції необхідна автоматизація збірки, тобто можливість автоматично компілювати програмне забезпечення і з'єднувати його в виконуваний файл. Це потрібно робити швидко, тому що великі збірки можуть вимагати багато часу. Без швидкого і надійного процесу складання не буде огляду, який необхідний для вирішення проблем інтеграції, що можуть виникнути, коли виконується процес збірки, вирішуються конфлікти між змінами, внесеними різними розробниками. У разі виявлення проблеми програміст, який працював над вирішенням конфлікту попередньої збірки,

може перевірити код за допомогою апаратного моделювання, не затримуючи інших розробників. Однак для досягнення такої ефективності процес інтеграції збірок повинен протікати безперервно. Тому, як тільки закінчується робота над попередньою збіркою, з'являється нова. Це значно відрізняється від щоденного або щотижневого випуску збірок, що практикується компаніями без системи безперервного розгортання [46].

Звичайно, для цього необхідна автоматизація процесу складання, тому що непрактично доручати людині завдання багаторазового створення збірок – знову і знову протягом усього робочого дня. Більш того, процес складання повинен виконуватися швидко, для чого часто необхідна багатопоточна збірка. При багатопоточній збірці, процес збору різних компонентів програмного забезпечення виконується паралельно, що прискорює весь процес. Для паралельного процесу розгортання потрібно більше обладнання і складніший сценарій. А чим складніше сценарій, тим корисніше стають інструменти управління складанням.

Основна ідея гнучкої розробки полягає в тому, що робота розкладається на невеликі, керовані фрагменти. Це є основною передумовою Continuous Integration: виправляти помилки рано і часто, що запобігає їх переростання в більші проблеми, які важче вирішити.

Одна з переваг цього методу – можливість випускати діючі версії меншого розміру, зібрані і протестовані багато разів у ході розробки проекту. Кожен випуск зменшує ризик для проекту, оскільки група перевіряє архітектуру, вимоги і оцінює терміни випуску. У гнучких методах ще не завершена робота називається чергою завдань (backlog). Коли починають розподіляти роботу малими кроками, так званими періодами, робота, відведена спринту, називається чергою завдань спринту (sprint backlog). Робота, що залишилася для майбутніх періодів, називається чергою завдань проекту. У період має включатися така кількість роботи, щоб її можна було виконати в терміни, відведений для спринту.

Цей процес залежить від ряду параметрів, так що група може точно передбачити кількість часу, яка буде потрібна для вирішення завдання, і, отже, кількість завдань, які можна відвести для одного спринту. Однак збір даних для оцінки цих показників – дуже трудомісткий процес, навіть для невеликої групи.

Коли такі невеликі групи об'єднуються, щоб зробити більш складний продукт, завдання стає занадто грандіозним, щоб його можна було вирішити власноруч.

На ринку існує багато інструментів, які допомагають організувати роботу, відслідкувати її завершення і визначити показники, пов'язані з тим, в якому обсязі, як швидко, як добре вона виконана. Якщо слідувати методам CI, то до цієї незавершеної роботи потрібно також швидко додавати виявлені помилки інтеграції і переносити її в початок списку високо пріоритетних робіт. Кращі продукти цього призначення на ринку забезпечують певний рівень інтеграції між новими елементами роботи і системою управління збірками, а тому помилки, виявлені в кожній збірці, можна швидко виправити і інтегрувати результат з готовими елементами роботи, підвищити його пріоритет і направити до потрібної групи.

В кінці кожної збірки запускаються автоматичні сценарії регресійних тестів. Це дозволяє розробникам негайно отримувати інформацію про помилки, знайдені в новій збірці. Цей крок попереджає розробників, коли новостворений ними код не відповідає вимогам. Без регресійних тестів розробники знають тільки, що складання виконано. Оскільки тести все одно повинні бути створені, то розробка через тестування не додає додаткової праці. Просто змінюється порядок дій – спочатку створюються тести, а потім код.

При традиційній розробці методом "водоспад" проект міг обходитися без всякої автоматизації тестування. Його можна було б описувати, виконувати і тестувати нескінченно цілою армією людей. Але як тільки починаються регулярні випуски, цей процес стикається з проблемами. Просто неможливо тестувати систему власноруч велику кількість разів на день.

Активне сповіщення розробників ведеться через такі служби як Mail, SMS, миттєві повідомлення у внутрішній системі компанії. Пасивне сповіщення проводиться за допомогою веб публікації і файлового сервера.

Одним з завдань даної роботи є вивчення та впровадження сервера безперервної інтеграції, який мав би такі характеристики:

1) на кожну зміну в репозиторії системи контролю версій збиралася б окрема збірка проекту;

2) автоматичний запуск модульних і інтеграційних тестів;

3) зворотній зв'язок з командою розробників в разі позитивного або негативного результату і зв'язок з менеджером завдань (Jira) для відстеження стану завдання і розгортання проекту;

4) інспекція вихідного коду, та надання детальних звітів розгортання та тесту проекту.

Основна мета безперервної інтеграції – забезпечити швидкий зворотний зв'язок. Ніщо так не підриває ідею безперервної інтеграції, як довге складання проекту. Що має вважатися "довгим" складанням. Більшість програмістів вважають збірку, триваючу цілу годину повністю невиправданою. Але є команди, які мріють про те, щоб прискорити складання.

Однак для більшості проектів, рекомендації десяти хвилинної збірки повністю задовольняють їх потреби. Більшість проектів досягають цієї мети. Це виправдовує себе тому, що кожна хвилина, яку виграє команда розробників в процесі складання, зберігає хвилину для кожного розробника у процесі тестування, кожен раз перед фіксацією змін. Так як практика безперервної інтеграції вимагає частих фіксацій, це економить багато часу.

Спостерігаючи за складанням, яке триває протягом години, робота по його оптимізації може здатися безперспективною. Такі проблеми можуть виникнути при роботі з новим проектом. Для корпоративних додатків, зазвичай вузьким місцем у процесі розробки є тестування, особливо ті з тестів, які використовують зовнішні сервіси, наприклад, тестування бази даних.

Ключовим моментом є налагодження поетапної збірки. Ідея поетапної збірки (також відомої як "послідовна збірка" – build pipeline) полягає в тому, що кілька збірок виконується послідовно. Фіксація змін викликає першу збірку (commit build). Збірка повинна виконуватися швидко і тому розробники повинні її скоротити, що має зменшити її здатність виявляти помилки. Основна ідея полягає в тому, щоб дотримуватися балансу між вимогами з виявлення помилок і швидкістю, щоб перша збірка (commit build) була достатня для того, щоб команда розробників могла продовжувати працювати.

Як тільки перша збірка буде досить надійна, щоб команда розробників могла впевнено працювати з кодом далі, можна запустити наступну, яка буде виконуватися значно довше і включати тестування, яке не увійшло до першої

збірку. Для цього можуть використовуватися додаткові сервери, так як тестування займає значно більше часу.

Як простий приклад візьмемо двоетапну збірку. На першому етапі проводиться компіляція і запускаються тести, які працюють тільки локально і не використовують базу даних. Такі тести працюють дуже швидко, утримуючи загальний час збирання в рамках 10 хвилин. Однак помилки інтеграції між модулями системи, особливо помилки, пов'язані з реальною базою даних не будуть виявлені. Збірка другого етапу запускає різні набори тестів, що працюють з базою даних і перевіряється більше інтеграційних аспектів системи. Другий етап може займати кілька годин.

За такого сценарію, розробники використовують перший етап в основному циклі безперервного розгортання. Збірка другого етапу запускається коли це можливо, використовуючи результати останньої успішної збірки першого етапу для початку тестування проекту. Якщо другий етап не збирається, це не означає, що вся робота зупиняється, але команда повинна домогтися виправлення виявлених помилок якомога швидше, зберігаючи збірку першого етапу в життєздатному стані. Незважаючи на те, що збірка другого етапу не зобов'язана завжди бути успішною, кожна виявлена у ній помилка повинна бути виправлена як мінімум протягом декількох наступних днів.

Якщо збірка другого етапу виявляє помилку, то це означає, що перша збірка повинна включати ще один тест. Настільки, наскільки це можливо, ви повинні бути впевнені в тому, що провал другої збірки призводить до появи нових тестів в першій збірці, щоб помилка була виправлена на першому етапі. Цей шлях приводить до збільшення обсягу тестування при першій збірці. Однак, існують випадки, коли неможливо використовувати швидке виконання тестів для перевірки такої помилки, в цьому випадку ви можете вирішити залишити цей тест в збірці другого етапу. На щастя, в більшості випадків цього можна уникнути.

Цей приклад демонструє двоетапну збірку, але описані вище основні принципи можуть використовуватися в схемах з будь-якою кількістю етапів. Складання після першої збірки можуть запускатися паралельно, наприклад, якщо у вас другий етап триває дві години, ви можете зменшити час відгуку збірки, розділивши обсяг тестування порівну на два комп'ютери. Використовуючи вторинні паралельні збірки, ви можете включити додаткові

види автоматизованого тестування, такі як тестування продуктивності, в свій звичайний процес [20, 36, 53].

## 1.2 Анализ переваг та недоліків систем безперервного розгортання проектів

Безперервна інтеграція проектів має ряд переваг, які наведені нижче:

1) скорочення ручних операцій – етапи створення, збирання і тестування програмного забезпечення проводяться в автоматичному режимі;

2) наявність робочої IT-системи протягом всього процесу розробки – у проектної команди завжди є свіжа версія рішення для демонстрації замовнику, отримання зворотного зв'язку і швидкого доопрацювання;

3) якість програмного забезпечення – в рамках Continuous Integration використовуються різні програмні засоби для контролю якості коду, що дозволяє скоротити кількість помилок;

4) мінімізація ризиків – дефекти виявляються на ранніх стадіях розробки програмного забезпечення, що допомагає уникнути збільшення термінів і вартості проекту;

5) окупність інвестицій в IT – автоматизація процесу розробки забезпечують високу ефективність і надійність інформаційної системи.

Нажаль системи безперервного розгортання мають ряд недоліків, які приведені нижче, хоча впровадження даної системи дає високий результат ефективності, що робить недоліки несуттєвим зауваженням:

1) витрати на підтримку роботи безперервної інтеграції;

2) потенційна необхідність у спеціальному навчальному сервері для потреб безперервної інтеграції;

3) негайний ефект від автоматизованого створення резервних копій неповного або непрацюючого коду перевчає розробників від виконання періодичних резервних включень коду в репозиторій;

4) проблеми інтегрування разі використання системи контролю версіями вихідного коду з підтримкою розгалуження, ця проблема може вирішуватися створенням окремої гілки (branch) проекту для внесення великих змін (код, розробка якого до працездатного варіанту займе кілька днів, але бажано частіше резервне копіювання в репозиторій). Після

закінчення розробки та індивідуального тестування такої гілки, вона може бути об'єднана з основним кодом або "стволом" (trunk) проекту [1, 75].

## 1.3 Проблеми та ризики пов’язані з системами безперервного розгортання проектів

Розглянемо проблеми, які виникають при розробці програмного забезпечення:

1) інтеграція програмного забезпечення складна робота;
2) багато рутинної роботи (збірка, тестування, розгортання);
3) необґрунтовані припущення про стан проекту;
4) тестувальники не можуть самостійно здійснювати збірку проекту;
5) розробникам важко, на першу вимогу, збирати і передавати проект тестувальникам;
6) постійний контроль середовища в якому відбувається збірка.

Основні ризики, які виникають при відсутності грамотної розробки програмного забезпечення:

1) зменшення прибутку внаслідок простою або низької продуктивності продукту. Цей ризик особливо актуальний для сучасних систем он-лайнової торгівлі, обробки замовлень, формування пакетів доставки і транспортної логістики;

2) необоротні логічні помилки в структурах зберігання даних в результаті несвоєчасних оновлень або ускладнених запитів на отримання актуальної інформації. Ризик набуває актуальності в умовах великого обсягу оброблюваних даних і великої кількості виконуваних операцій за одиницю часу. Критичний цей ризик і для систем оперативного контролю та моніторингу систем, що працюють в режимі реального часу, систем автоматизованого управління агрегатами і апаратами, а також для систем життєзабезпечення;

3) необоротні зміни або втрата операційних даних через збій програми, що викликаний надмірними навантаженнями. Ризик актуальний в сфері зберігання та обробки фінансової інформації, яка носить унікальний характер, не може бути відновлений на основі емпіричних припущень і несе пряму загрозу фінансових збитків. Крім фінансових систем, це становить

небезпеку для проектів білінгу і реєстрації процесингової інформації. Даний ризик актуальний не стільки через погрози реальних втрат даних, скільки за рахунок витрат часу, який потрібен на відновлення інформації. Якщо інформація не може бути відновлена або зареєстрована в системі в результаті збою, ризик стає критичним.

Основними пунктами економії при розробці програмного забезпечення є:

1) економія коштів на реалізацію зайвої надійності продукту. Витрати пояснюються витратами на реалізацію зайвої функціональності, розрахованої на виконання явно завищених (часом навіть в кілька разів) вимог до продуктивності, або на зайву оптимізацію архітектури та коду, що збільшує терміни розробки і тестування, а отже, підвищує вартість проекту в цілому;

2) зменшення витрат на реалізацію зайвої надійності інфраструктури. Дані витрати за своїм характером і з причин виникнення схожі з описаними вище і виникають через недостатню інформацію про можливості застосовуваних технологій та про свідомо завищених, тобто песимістичних результатах емпіричних прогнозів. Наслідки цього виражаються у витратах на аналіз і реалізацію зайвої надійності інфраструктури, механізмів дублювання логіки і дорогих рішень по агрегації та міграції даних;

3) економія часу та коштів на виявлення та усунення помилок під час експлуатації. Сюди відносяться статті витрат служб експлуатації та контролю, а також системного адміністрування на локалізацію та виявлення причин збою, на відновлення систем до робочого стану, а також на проведення робіт по відновленню даних і стану продуктивних систем [11].

## 1.4 Огляд існуючих систем безперервного розгортання

Було проведено аналіз існуючих систем безперервної інтеграції, вивчені їх можливості. Особливу увагу привертають такі відомі продукти, як:

1) Team City;
2) Cruise Control;
3) Hudson.

На рисунку 1.8 наводиться співвідношення популярності продуктів на ринку IT. Як видно на ньому, популярним є Jenkins / Hudson.

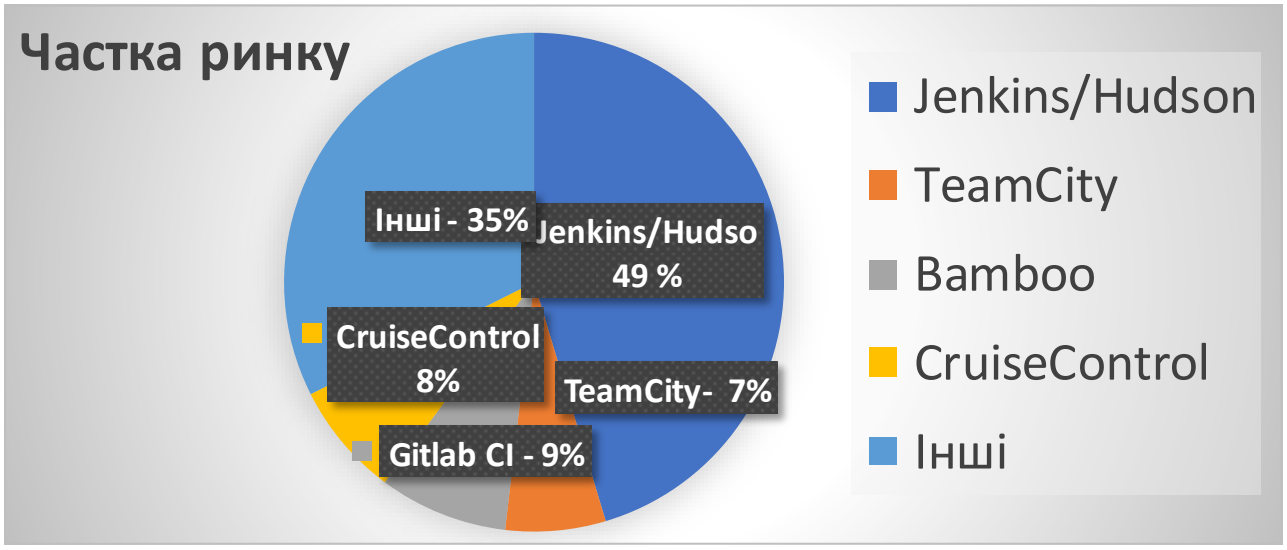


Рисунок 1.8 – Співвідношення популярності систем безперервного розгортання на ринку IT

Данні порівняння основних характеристик трьох популярних систем безперервної інтеграції наведено у таблиці 1.1, розглянемо більш детально кожен продукт.

Система безперервної інтеграції Jenkins / Hudson – це безкоштовний, розроблений на платформі Java EE продукт з відкритим вихідним кодом, що розробляється власним співтовариством. Jenkins де-факто є стандартом безперервної інтеграції. Він поставляється в форматі Java Web Archive архіву і може бути запущений як окремим процесом, так і розгорнуто в контейнері сервлетів. Запускається в контейнері сервлетів, таких як Apache Tomcat або GlassFish. Підтримує інструментарій для роботи з різними системами контролю версій включаючи CVS, Subversion, Git і Clearcase, може збирати проекти Apache Ant і Apache Maven, а також виконувати shell скрипти і команди.

Таблиця 1.1 – Порівняння основних технічних характеристик

| Назва | Плат-форма | Ліцензія | Збирачі для Windows | Java збирачі | Інші збирачі | Повідомлен-ня | IDE інтеграція | Інші інтеграції |
|---|---|---|---|---|---|---|---|---|
| Team City | Сер-влет кон-тейнер | Пропри-єтарна | MSBuild, NAnt, Visual Studio | Ant, Maven, IDEA Inspect ions, Gradle | Rake, FxCop, командний рядок | Email, XMPP, RS S, IDE, SysTray | Eclipse, Visual Studio, IntelliJ IDEA, RubyM ine, PyCharm, PhpStorm, WebStorm | Jetbrains Youtrack, JIRA, Bugzilla, FishEye FindBugs, PMD, dotCover, NCover |
| Cruise Control | Крос-сплат-форменний | BSD-style | MSBuild, NAnt, Visual Studio | Невідомо | Командний рядок | E-mail, CCTray, RSS | Невідомо | Невідомо |
| Hudson / Jenkins | Сервлет контей-нер | Creative Commons та MIT | MSBuild, NAnt | Ant, Maven 2, Kundo | Cmake, Gant, Gradle, Grails, Phing, Rake, Ruby, SCons, Python, шелл скрипт і командний рядок | Android, Email G oogle Calendar, IRC, XMPP, RSS, Twitter | Eclipse, IntelliJ IDEA, NetBeans | Bugzilla, Google Code, JIRA, Bit-bucket, Redmine, Find Bugs, Checkstyle, PMDand Mantis, Trac |

Сервер безперервної інтеграції Jenkins (рисунок 1.9) дозволяє регулярно проводити збирання програмного забезпечення. Можна призначати початок складання на певні моменти часу або на певні події. Наприклад, поява в репозиторії нового коду від команди розробників або як тригер успішного збору іншого програмного продукту. Фактично система представляє собою виконання одного або декількох "кроків збірки" з певним набором параметрів. Кожен "крок збірки" представляє собою команду. Команди можуть бути чотирьох типів: Cmd, Shell, Ant, Maven [29, 31].

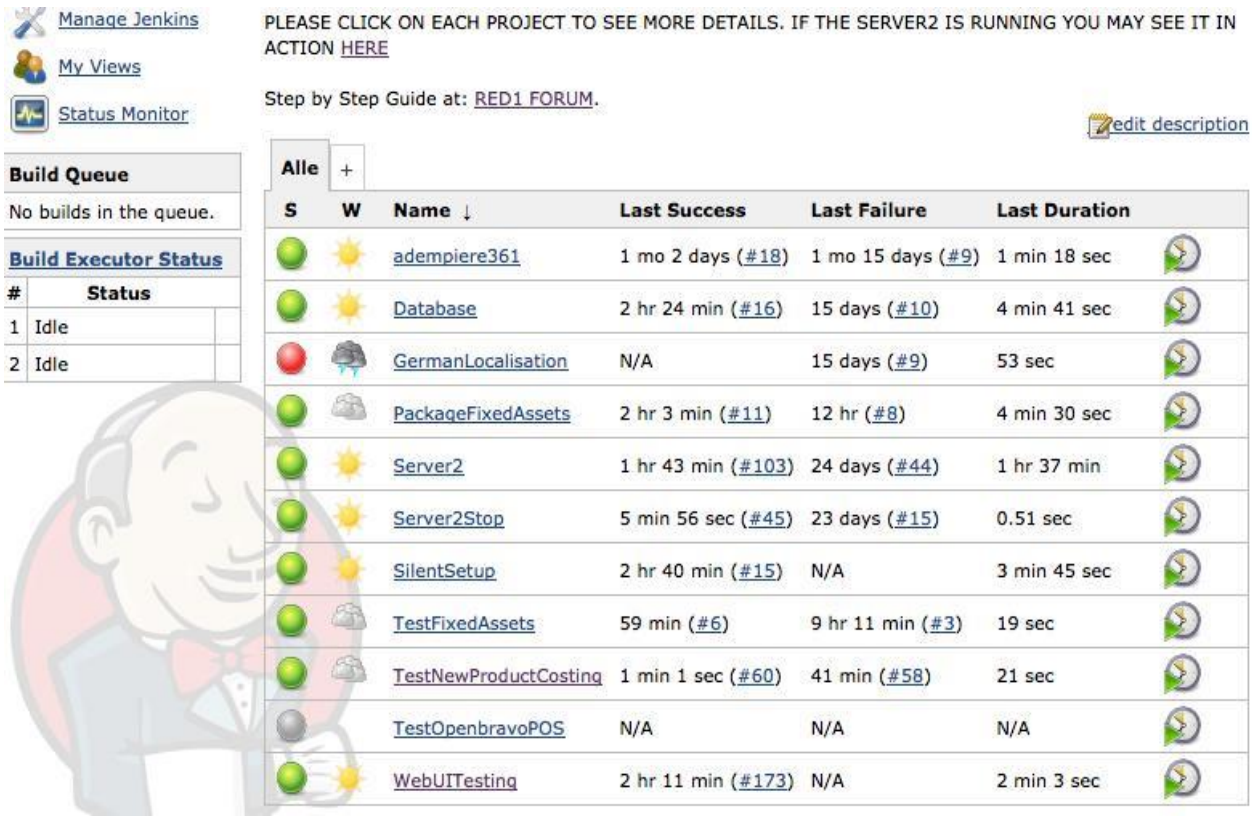


Рисунок 1.9 – Веб-інтерфейс системи безперервного розгортання Jenkins

Таким чином, можна виконувати не тільки окремі інструкції командного рядка, а й цілі скрипти. Така зручна і гнучка система дозволяє запускати різні тести, що допомагають здійснювати контроль якості розроблюваного програмного продукту.

Jenkins надає можливість запускати кілька збірок паралельно. Для цього використовується розподілена архітектура. Машина, на якій

запущений сам Jenkins, називається master. Машини, на яких запущений Jenkins агенти, називаються slave.

Така модель дозволяє одночасно збирати безліч великих проектів, розподіляючи навантаження на різні машини. Це дає можливість працювати з різними версіями і конфігураціями кінцевих продуктів, що, зрозуміло, зменшує витрачання часу і збільшує продуктивність відділів розробки та тестування.

Jenkins має дуже зручний веб-інтерфейс і досить легко розширюється. Для того, щоб додати в Jenkins новий функціонал необхідно розробити плагін, який реалізує даний функціонал. На даний момент випущено більше трьохсот плагінів до Jenkins.

CruiseControl – інструмент безперервної інтеграції програмного забезпечення на платформі Java, націлений на автоматизацію процесу складання. Управління та перегляд збірки здійснюється через веб-інтерфейс. Інтегрується з Apache Ant, різними системами управління версіями.

CruiseControl є відкритим програмним забезпеченням, поширюється під BSD-подібною ліцензією. Спочатку він був створений співробітниками компанії ThoughtWorks (включаючи Мартіна Фаулера) з метою забезпечення безперервної інтеграції для одного з проектів, пізніше інструмент був виділений в окремий додаток [48, 57].

Цикл збірки (Build loop) в інструменті реалізований як демон, періодично перевіряє систему управління версіями на зміни в кодової базі, що виробляється в разі необхідності складання та публікує повідомлення про її статус.

Для звітності про статус збірки використовуються два методи – перший (класичний) реалізований у вигляді стандартних JSP-сторінок, другий (починаючи з версії 2.7) – відображає результати на приладовій панелі (Dashboard), на якій скомбінована велика кількість різних уявлень та широко використовуються кольори, значки, елементи, спливаючі при наведенні об'єкти.

Крім Java-версії існують варіанти інструменту для платформи Microsoft .Net (CruiseControl.NET, CCNet) і версія для Ruby-середовищ (Cruise Control.rb).

Система безперервного розгортання проектів Cruise Control має наступні можливості:

1) працює за розкладом або стежить за змінами:
   а) в репозиторії систем контролю версій;
   б) в файлах і директоріях;
   в) в збірці інших проектів;
2) вміє витягувати нову версію з систем контролю версій і збирачі, використовуючи Ant, Maven, скрипт;
4) публікація результатів: в залежності від складання на веб-сервері;
5) розсилає пошту;
6) виконує скрипти (shell, bat);
7) виконує скрипти Ant;
8) конфігурується одним XML файлом – його теж можна зберігати у системі контролю версій.

На рисунку 1.10 представлений веб-інтерфейс сервера безперервної інтеграції Cruise Control.

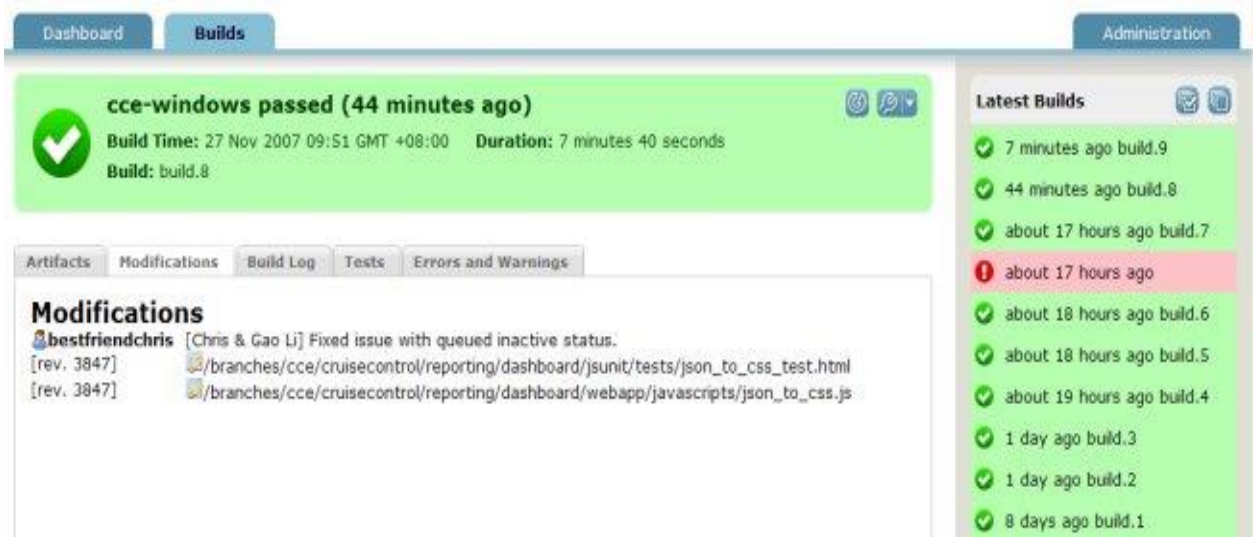


Рисунок 1.10 – Веб-інтерфейс серверу безперервної інтеграції Cruise Control

Система безперервної інтеграції TeamCity – це багатофункціональна система інтеграції, готова до роботи відразу ж після інсталяції. Вона підтримує безліч систем контролю версій, автентифікації, збірки та тестування прямо з коробки. При цьому TeamCity легко розширюється: для багатьох операцій навіть не потрібно знати мову програмування Java.

Система безперервного розгортання TeamCity має такі можливості:

1) миттєві повідомлення про помилки збірки. Можливість запускати збірку і тестування зміненого коду без коммітів в систему контролю версій, прямо з IDE;

2) вбудована підтримка Ruby і XCode проектів;

3) статистичні звіти за результатами збірок можуть задовольнити навіть найвибагливішого користувача;

4) легке управління фермою білд-агентів, включаючи їх автоматичне оновлення, розбиття на пули і звіти по завантаженню;

5) управління загальними ресурсами, що дозволяє без проблем обмежувати доступ до спільного використання баз даних, тестових пристроїв тощо;

6) підтримка змішаної автентифікації, що дозволяє використовувати різні способи автентифікації (LDAP, Windows Domain, вбудована) одночасно;

7) більше 100 безкоштовних готових до використання плагінів [30].

На рисунку 1.11 представлений веб-інтерфейс сервера безперервної інтеграції Team City.

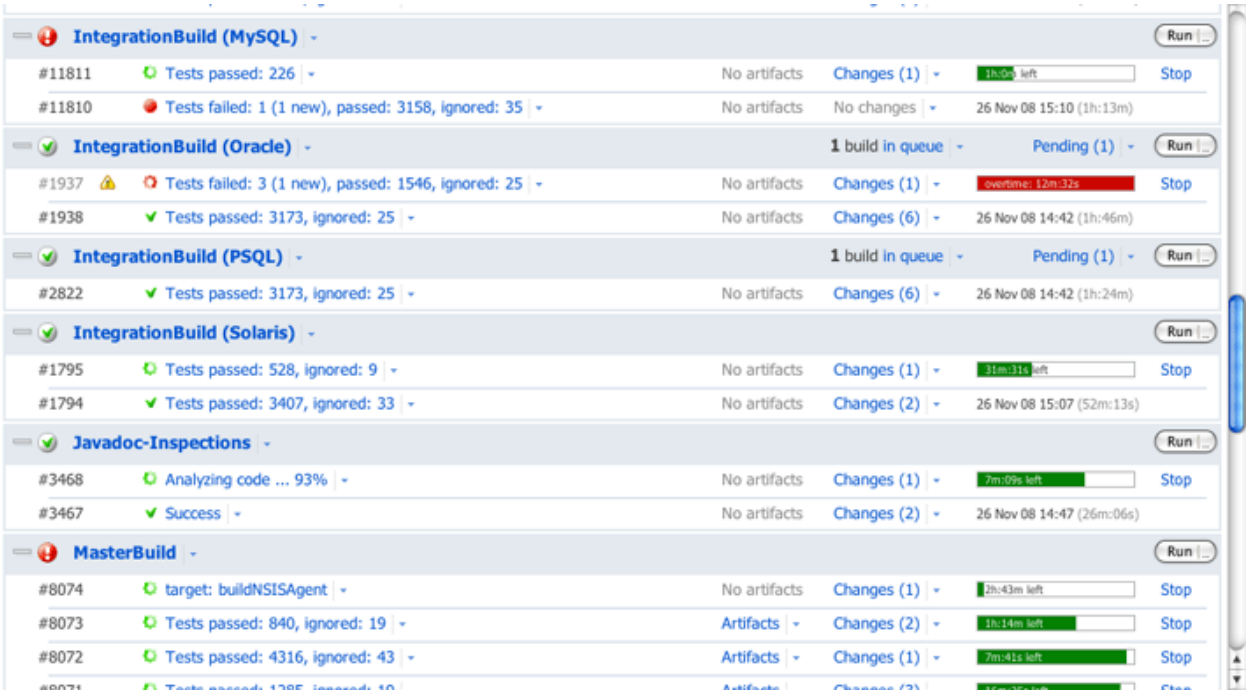


Рисунок 1.11 – Веб-інтерфейс системи безперервного розгортання Team City

## 1.5 Поняття та сутність конкурентоспроможності

Конкурентоспроможність об’єктів – це здатність певних об’єктів чи суб’єктів перевершити конкурента у заданих умовах. Конкурентоспроможні об’єкти можна розбити на чотири групи:

1) товар;

2) підприємство, як виробник товарів;

3) галузь, як сукупність підприємств, що пропонує товари чи послуги;

4) регіони, області, країни або їх групи.

Аналізуючи визначення конкурентоспроможність можна виділити наступні ознаки даного поняття (рисунок 1.12) [13].


Рисунок 1.12 – Класифікація конкурентоспроможності за певними ознаками

Конкурентоспроможність з'являється в ринкових умовах, коли на ринок виходить кілька виробників аналогічного товару. Щоб залишатися на ринку конкурентоспроможним підприємством необхідно, щоб продукт задовольняв потреби покупця в даний момент і був привабливим у порівнянні з аналогічними виробами завдяки кращій якості.

Конкурентоспроможність підприємств – категорія кількісно вимірювана, що вимагає застосування методів економіко-математичного аналізу, модельного апарату, здатного з більшою точністю відобразити існуючий стан і дати достовірний прогноз розвитку на перспективу. Під рівнем конкурентоспроможності підприємства слід розуміти, відносний показник, що відображає характеристику його можливостей протистояти конкурентному натиску на конкретних ринках (окремих сегментах) в певний період часу. При цьому методи його встановлення, незважаючи на традиційний набір обчислювальних процедур, можуть істотно відрізнятися. Зокрема, відмінності часто спостерігаються за складом локальних (приватних) показників, що значною мірою залежить від галузевої спеціалізації підприємства, масштабів виробництва, організаційної структури використання інструментів маркетингу і т. д [28, 38].

Починати дослідження і оцінку конкурентоспроможності підприємства слід з аналізу стану його місця і ролі в конкурентному середовищі. При цьому важливо знати не тільки частку ринку займаних сегментів, а й чи відбуваються в них зміни. Також необхідно чітко уявляти ступінь популярності даного підприємства (ставлення до нього споживачів, встановити мотиви переваг при виборі, того, чи іншого виробництва при покупці продукції, включаючи якість, рівень цін, дизайн, упаковку і т. д.). За результатами моніторингу проводиться зіставлення показників з аналогічними характеристиками конкуруючих суб'єктів і виконуються прогнозні розрахунки на майбутнє. При широкій номенклатурі виробів чи послуг необхідно весь асортимент продукції розбити по товарними групами, ґрунтуючись на ряді критеріїв (вид продукції, упаковки, ціна, клас і т. д.). Тут також важливо враховувати динаміку цього показника.

В умовах ринку нереально домогтися стабільного успіху в бізнесі, якщо не планувати його ефективний розвиток, не акумулювати інформацію про власні перспективи і можливості, про стан цільових ринків, про становище конкурентів і своєї конкурентоспроможності. Конкурентоспроможність підприємства – це відносна характеристика, яка виражає ступінь відмінності даного підприємства від конкурентів в сфері задоволення потреб клієнтів.

Висока конкурентоспроможність підприємства обумовлюється задовільністю і готовністю споживачів повторно придбати продукцію цієї

фірми, відсутність претензій до підприємства з боку суспільства, акціонерів, партнерів, престижністю роботи на даному підприємстві.

Конкурентоспроможність підприємства, будучи багатогранним поняттям, не тільки включає якісні і цінові параметри промислової продукції, але й залежить від рівня менеджменту, сформованої системи управління фінансовими потоками, інвестиційного та інноваційного механізму його діяльності. Крім того, на конкурентоспроможність впливає кон'юнктура, що складається на тому чи іншому ринку, ступінь конкуренції, яку відчуває підприємство з боку інших учасників ринку, технічна оснащеність, ступінь впровадження інновацій, мотивація і кваліфікація персоналу, фінансова стійкість. За інших рівних умов найважливішу роль набуває маркетингова складова конкурентоспроможності підприємства. Маркетингова орієнтованість на виявлення найбільш значущих потреб клієнтів, облік змін споживчих переваг, оцінку перспективності сегментів товарів ринку, розробку і втілення дієвих стратегій щодо підвищення конкурентоспроможності. Необхідною умовою визначення конкурентоспроможності виступає наявність конкуренції.

Конкуренція здійснюється на ринку в умовному місці купівлі-продажу конкретного виду товару, укладання торговельних угод, які здійснюються в певних по інтенсивності умовах конкуренції з дотриманням етичних і правових норм і правил [3, 55].

Конкурентоспроможність товару – здатність відповідати вимогам конкурентного ринку, запитам покупців в порівнянні з іншими аналогічними товарами, представленими на ринку. Вона визначається, з однієї сторони, якістю товару, його технічним рівнем, споживчими властивостями, з іншого боку – цінами, що встановлюються продавцями товарів. Крім того, на конкурентоспроможність впливають мода, продажний і після продажний сервіс, реклама, імідж виробника, ситуація на ринку, коливання попиту.

Таким чином, конкурентоспроможність товару – ринкова характеристика, сукупність його переваг на ринку, що сприяють успішній реалізації в успішній конкуренції. Дане поняття визначається системою технічних, споживчих і економічних показників: технічним рівнем продукції, функціональними, соціальними, естетичними і іншими корисними властивостями, ціною покупки і витратами на споживача.

Оцінка проводиться шляхом порівняння цих показників з товаром-конкурентом, перспективними зразками та нормативами. У різних авторів у визначеннях конкурентоспроможності по важливості лідирує якість, корисність, співвідношення ціна-якість, привабливість, здатність до реалізації, вміння витримувати конкуренцію і т.д. Нам видається, що конкуренто-спроможність – комплексний показник товару, результат роботи всіх елементів маркетингу, що забезпечує конкурентні переваги на ринку [3, 5, 26].

# 2 АНАЛІЗ ПОКАЗНИКІВ КОНКУРНТОСПРОМОЖНОСТІ

## 2.1 Аналіз методики оцінки конкурентоспроможності

У діяльності сучасних підприємств далеко не останню роль почало відігравати поняття конкурентоспроможності.

Під конкурентоспроможністю підприємства прийнято розуміти його здатність бути затребуваним і успішним на ринку, змагатися з конкуруючими фірмами і отримувати більше економічних вигід у порівнянні з компаніями-постачальниками послуг у сфері розробки програмного забезпечення.

В цілому характеристика комплексна і вона може бути виражена через набір показників. Для визначення положення, займаного економічним суб'єктом на внутрішньому і зовнішньому ринку необхідно проводити оцінку його конкурентоспроможності.

Оцінка конкурентоспроможності, яку необхідно проводити компаніям, часто буває заснована на інтуїтивних відчуттях, однак, її цілком можна формалізувати, описавши колом показників, які, дозволяють здійснити саму оцінку і дозволяють виділити напрямки підвищення конкурентоспроможності через виявлення факторів, що впливають.

Показники, які можуть бути використані при оцінці конкурентоспроможності компанії, різні і їх набір може відрізнятися в залежності від використовуваної методики оцінки.

У сучасній науці існує шість основних підходів до визначення конкурентоспроможності.

Відповідно до першого підходу конкурентоспроможність розглядається з точки зору переваг в порівнянні з конкурентами.

Другий підхід заснований на теорії рівноваги А. Маршала. У постачальника послуг немає приводу для переходу в інший стан, і він досягає максимального прибутку і рівня збуту.

Третій підхід полягає в оцінці конкурентоспроможності за якістю створюваної продукції на основі аналізу за різними характеристиками компетентності.

Четвертий підхід являє собою матричну методику оцінки конкурентоспроможності, реалізується за допомогою складання матриць і попередньому виборі стратегії.

П'ятий підхід структурний, відповідно до нього положення підприємства можна оцінити через такі показники як: рівень монополізації галузі, наявність бар'єрів для нових підприємств, що з'являються на ринку.

Шостий підхід функціональний, його представники визначають співвідношення між витратами і ціною, обсяги завантаження трудових ресурсів, кількість створюваної продукції, що розробляється та інші показники. Відповідно до цього підходу конкурентоспроможними вважаються компанії, в яких краще налагоджено процес створення продукції, більш ефективно поставлено управління фінансовими ресурсами. Наприклад, цей підхід застосовується в компанії "Дан енд Бредстріт", це відома американська консультаційна фірма [24, 38].

На сьогоднішній день розроблено безліч методів оцінки конкурентоспроможності підприємств, їх можна класифікувати таким чином:

1) матричні методи оцінки;

2) методи, які базуються на проведенні оцінки конкуренто-спроможності створюваної продукції;

3) методи, які базуються на теорії ефективної конкуренції;

4) комплексні методи оцінювання конкурентоспроможності.

Матричні методи оцінки є досить простими і дають точну інформацію. Більш того вони засновані на розгляді процесу конкуренції в розвитку і в разі наявності правдивої інформації дають можливість здійснити досить якісний аналіз конкурентних позицій.

Методи, які базуються на проведенні оцінки конкурентоспроможності продукції, пов'язують за допомогою поняття "ефективного споживання". Вважається, що конкурентоспроможність підвищується, якщо вище якість

товару і менше його вартість. Серед позитивних рис цих методів можна назвати: простоту проведення оцінки. Але разом з тим вони не дають повного уявлення про сильні та слабкі сторони в роботі підприємства.

Методи, які базуються на теорії ефективної конкуренції. Відповідно до цього методу, найбільш конкурентоспроможними вважаються фірми, в яких найкращим чином налагоджена робота всіх підрозділів і служб. Оцінка ефективності роботи будь-якої такої структури має на увазі оцінку ефективності використання нею ресурсів. Така методика оцінювання застосовується найбільше в оцінці IT компаній і включає всі найважливіші оцінки бізнес процесів, виключаючи дублювання конкретних показників, дає можливість створити загальну картину конкурентного становища фірми на внутрішньому і зовнішньому ринку швидко і точно.

Комплексні методи оцінювання конкурентоспроможності підприємства здійснюються з використанням методу інтегральної оцінки. Цей метод включає дві складові: по-перше, критерій, що характеризує ступінь задоволення потреб споживача, по-друге, критерій ефективності виробництва. Позитивною рисою цього методу можна назвати простоту здійснюваних розрахунків і можливість однозначно інтерпретувати результати. Разом з тим, важливим недоліком є неповна характеристика діяльності підприємства [55].

## 2.2 Аналіз методик оцінки конкурентоспроможності на основі концепції ціннісного ланцюга М. Портера

Основою методики оцінки конкурентоспроможності є запропонована Портером концепція ціннісного ланцюга (Value Chain), де всі види діяльності фірми поділяються на дві категорії: основну діяльність (постійне виробництво, збут, доставка і обслуговування товару) і підтримує діяльність (забезпечення компонентами виробництва: технологією, людськими ресурсами, інфраструктурою). Узагальнений ціннісний ланцюг фірми

представлений на рисунку (рисунок 2.1).

Будь-яка фірма представляє собою суму дій необхідних для проектування, виробництва, ринкового просування, доставки та обслуговування продукції. Ціннісний ланцюг фірми є відображенням її історії, стратегії, підходів до здійснення стратегії, а також способів виконання, закладених в ціннісний ланцюг дій.

Тут слід зробити доповнення з приводу термінів, термін цінність розуміється як сума, необхідна для виконання тієї чи іншої дії, проте це не тільки вартісний показник, але і показник якості, значення виконуваної дії для досягнення конкурентної переваги. Оскільки не всякий показник конкурентоспроможності можна висловити грошовою сумою, використання терміну ціннісний, а не вартісний, ланцюг є кращим.


Рисунок 2.1– Узагальнений ціннісний ланцюг підприємства

Ціннісні дії і прибуток показують ціну товару чи послуги, пропонованої фірмою. Однак, ціннісний ланцюг фірми, це не ще один спосіб визначення собівартості, сенс його в іншому. Він призначений для визначення сильних і слабких сторін підприємства, використовуючи його, можна побачити реальні і потенційні джерела конкурентної переваги, а також оптимізувати роботу фірми, відповідно до обраної конкурентної стратегії.

Кожну ціннісну дію використовує: людські ресурси і технологію, що

дозволяє виконувати покладені на неї обов'язки; крім цього, використовується і створюється та чи інша інформація, наприклад, технічні специфікації, прототипи, дані про потенційних покупців та інше.

Усі ціннісні дії можуть бути розділені на 2 великі групи: основну і підтримуючу діяльність. До основної діяльності відносяться дії пов'язані з фізичним створенням продукту, його продажем і сервісним обслуговуванням. У будь-якій фірмі основна діяльність може бути розділена на 5 категорій (рисунок 2.1). Підтримуюча діяльність повинна забезпечувати основну, надаючи технологію, управління, і виконання функцій, пов'язаних з підприємством в цілому. Інфраструктура фірми не пов'язана з конкретними основними діями, її функція в забезпеченні ефективних внутрішньо фірмових зв'язків і взаємодії [13].

Основна діяльність і підтримує ціннісні дії, є будівельними блоками конкурентної переваги. Джерела переваги в витратах можуть перебувати як у виробничому блоці, так і в блоці вихідної логістики. Концепція ціннісного ланцюга дозволяє побачити місце освіти конкурентної переваги, дозволяє намітити стратегічні плани зосередження зусиль фірми на конкретні блоки внутрішньо-фірмового простору, а також безпосередньо знайти місця неефективного використання капіталу і шляхи виходу з кризи.

Основна діяльність фірми може бути розбита на 5 категорій, кожна з яких також складається з властивих їм дій:

1) вхідна логістика. Дії, пов'язані з отриманням, зберіганням і розповсюдженням по підприємству інформації, такі як: стадії розробки, контроль за термінами, планування термінів, зворотній зв'язок з клієнтом;

2) створення продукту. Дії, пов'язані з переробкою вхідної інформації у кінцевий продукт, такі як: аналіз технічного завдання, збірка проекту, обробка і перевірка;

3) вихідна логістика. Дії, пов'язані з прийманням продукту замовником;

4) маркетинг і продаж. Дії, спрямовані на стимулювання покупців і безпосереднього продажу готових продуктів компанії, такі як: реклама, просування продукту, маркетинг, ціноутворення;

5) сервіс. Дії, спрямовані на відновлення і підтримання цінності, придбаного покупцем продукту, такі як: інсталяція, усунення помилок, підготовка обслуговуючого персоналу, надання допоміжних модулів.

Кожна з цих категорій дій може бути надзвичайно важливою для отримання конкурентної переваги. Вони присутні в кожному підприємстві, граючи при цьому різні ролі, маючи різну важливість, в залежності від галузі, в якій діє фірма.

Підтримуюча діяльність, показана на рисунку 2.1 вище, може бути розділена на 4 загальних категорії:

1) постачання. Постачання покликане забезпечити фірму необхідними для її функціонування матеріалами, машинами, і іншими товарами. Часто постачання асоціюється з основною функцією фірми, такий як вхідна логістика, однак постачання обслуговує весь ціннісний ланцюг фірми, наприклад, забезпечує обладнанням виробничий блок (програмісти), офіс-менеджмент та інше. Як і всі інші дії у постачання є власна технологія, що включає в себе: інформаційну систему, кваліфікаційні та виробничі вимоги. Постачання грає важливу конкурентну роль, хоча частка витрат на саму службу постачання мала, проте її якісні характеристики грають дуже важливу роль для отримання конкурентної переваги, причому як для лідерства в витратах, так і для диференціації;

2) розвиток технології. Кожна ціннісна дія включає в себе технологію, ноу-хау, процедури, або технології створення товару. Кількість технологій, що використовуються на підприємствах дуже велика починаючи з технологій підготовки документів на постачання товарів, закінчуючи сервісної або виробничої технологією. Більш того, багато ціннісних дій використовують технологію, яка включає в себе кілька субтехнологій. Розвиток технологій має безліч форм, наприклад, потрібно розвинути дизайн продукту, і в той же час потрібні нові технології маркетингових досліджень,

інтерактивна система реєстрації замовлень. Тому, використовувана технологія, є найважливішим фактором конкурентної переваги, причому необхідно враховувати, що важлива не тільки виробнича технологія, але і технології виконання інших основних ціннісних дій;

3) управління ресурсами розробки – включає в себе дії пов'язані з наймом, підготовкою, контролем та оплатою всіх типів персоналу. Управління персоналом обслуговує як основну, так і допоміжну діяльність. Персонал значно впливає на конкурентну перевагу, і можливо навіть грає головну роль. Для переваги на диференціації важлива кваліфікація і творчий потенціал робочої сили, в той же час лідерство у витратах в деяких галузях може бути досягнуто тільки за допомогою дешевої робочої сили;

4) інфраструктура фірми – складається з дій, що забезпечують функціонування фірми в цілому, це: управління вищої ланки, планування, фінанси, бухгалтерський облік, зв'язки з урядом. На відміну від інших ціннісних дій інфраструктура не обслуговує окремі категорії, а працює на всю ціннісну ланку підприємства. Вона також може бути джерелом конкурентної переваги.

Методика аналізу конкурентоспроможності передбачає використання ціннісного ланцюга підприємства в якості основи для конкурентного аналізу. Це обумовлено: по-перше, універсальністю концепції ціннісного ланцюга, можливістю його застосування в будь-якій галузі; по-друге, побудова ціннісного ланцюга підприємства дозволяє побачити існуючі та потенційні джерела конкурентної переваги; по-третє, ціннісний ланцюг в графічному вигляді демонструє переваги і недоліки, досліджуваного підприємства, а також дозволяє визначити рейтинг кожного ціннісного блоку [38, 57].

Позитивним моментом даної методики є її гнучкість, тобто можливість деякого доопрацювання під потреби конкретного підприємства або галузі. Це пояснюється тим, що при складанні загальної методики, врахувати всі джерела конкурентної переваги характерні для кожної фірми і (або) галузі.

Експертній групі слід відповісти на питання, згруповані за ціннісними

блоками підприємство, при цьому, оцінка 1 є низькою, тобто твердження або питання не відповідають реальному стану справ, оцінка 3 – часткова відповідність, 5 – повна відповідність, найвища оцінка. Відповідати на питання слід обґрунтовано, з більшою часткою применшення власних можливостей, при аналізі своєї фірми, і перебільшення можливостей конкурентів [66, 70].

Проведемо оцінку конкурентоспроможності підприємства “NEKWALL” використовуючи розроблений метод.

Основна діяльність підприємства базується на розробці комерційного програмного забезпечення на основі веб технологій. Проведемо табличну оцінку конкурентоспроможності усіх процесів, представлених на рисунку (рисунок 2.1).

### 2.2.1 Основна діяльність підприємства

Вхідна логістика визначає ціннісні дії, пов’язані з початковою стадією роботи з клієнтом. Проводиться оцінка взаємин з клієнтом, визначаються умови розробки, визначається команда проекту та її кваліфікаційний рівень.

Налагоджена система взаємодії з клієнтом є однією з найважливіших речей, які впливають на рівень конкурентоспроможності. Тому при проведенні оцінки поточного стану вхідної логістики, значну увагу слід приділити саме ціннісним діям, пов’язаним з взаємодією з клієнтом. Ціннісні дії вхідної логістики наведені у таблиці 2.1.

Таблиця 2.1– Вхідна логістика

| Ціннісні дії | Оцінка 1,2,3,4,5 |
|---|---|
| 1 | 2 |
| Ефективність обробки замовлень (враховувати забезпеченість спеціальним обладнанням, навички персоналу). | 3 |

Продовження таблиці 2.1

| 1 | 2 |
|---|---|
| Використовується система автоматизації обробки нових замовлень клієнта. (Використовується, не використовується, використовується частково) | 4 |
| Оцініть розмір витрат на зберігання продукції (хостинг, хмарні дата центри) | 3 |
| Умови розробки продукту відповідають вимогам конкурентної стратегії (терміни, швидкість, вартість) | 2 |
| Кваліфікація персоналу по роботі с клієнтом | 3 |
| Оцініть ваші взаємини з клієнтами | 5 |
| Оцініть кількість персоналу | 4 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 4 |
| **Разом середня оцінка:** | 3,5 |

Виробництво включає в себе ціннісні дії, пов’язані саме з процесом розробки програмного забезпечення. У даному блоці проводиться оцінка поточного стану обладнання для роботи над проектом, оцінюється рівень використання технологій проектів, рівень кваліфікації персоналу, включених до проектів, відповідність дизайнерського погляду з сучасними трендами. Також у даному блоці ми можемо виділити такі ціннісні дії, як план розвитку, програми покращення якості розробки. Якщо підприємство займається розробкою внутрішніх проектів, то також розглядається якість упаковки створюваного продукту та розмір виробничих витрат на створення даних продуктів. Ціннісні дії виробництва наведені у таблиці 2.2.

Таблиця 2.2 – Виробництво

| Ціннісні дії | Оцінка 1,2, 3,4,5 |
|---|---|
| Устаткування відповідає сучасним вимогам (гірше, на тому ж рівні або краще, ніж у конкурентів) | 5 |
| Технологія розробки програмного забезпечення, (відповідає вимогам часу, передова, відстала) | 3 |
| Рівень кваліфікації працюючих | 4 |
| Оцініть аналіз розміру обсягу майбутньої розробки | 4 |
| Оцініть кількість працюючих | 3 |
| Яка якість продуктів, що випускаються (бажано порівнювати з продукцією конкурентів) | 3 |
| Чи відповідають цілі керівників і цілі вашого підприємства (не відповідають, відповідають, відповідають частково) | 3 |
| Оцініть виробничі нововведення у Вашому підприємстві (їх кількість, якість, обсяг інвестицій) | 2 |
| Чи є план розвитку виробництва (немає, є, як виконується) | 3 |
| Чи є програма підвищення якості продукції | 4 |
| Ваше виробництво здатне перебудовуватися на інші технології? | 3 |
| Наскільки упаковка сприяє продажам, привертає увагу? | 3 |
| Оцініть дизайн продукції, наскільки він відповідає сучасним смакам споживачів | 3 |
| Розмір виробничих витрат (оптимальні, можуть бути знижені, високі) | 4 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 4 |
| **Разом середня оцінка:** | 3,4 |

Вихідна логістика є ключем фінансової ланки, адже саме на цьому

етапі продукт передається замовнику, та замовник проводить виплати по зробленій праці. На даному підрозділі можна виділити такі ціннісні дії, як вихідна документація для продукту (інструкція використання), розглядається розміщення створюваного продукту у дата центрах компанії та витрати на їх утримання, визначається технічна підтримка, та оцінюється взаємодія з клієнтом. Якщо створюваний продукт є внутрішнім для підприємства, то визначається фізичне поширення товару (наприклад, як додаткова послуга для створюваного продукту на замовлення). Ціннісні дії вихідної логістики наведені у таблиці 2.3.

Таблиця 2.3 – Вихідна логістика

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Оцініть рівень створення документації готового продукту | 4 |
| Умови зберігання готового продукту у дата центрах підприємства | 4 |
| Оцініть витрати на зберігання готового продукту | 3 |
| Оцініть чисельність персоналу технічної підтримки | 4 |
| Які Ваші зв'язку з покупцями, після завершення проекту | 5 |
| Фізичне поширення товару | 5 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 4 |
| **Разом середня оцінка:** | 4,1 |

Маркетинг та продажі. У випадку ІТ компанії, даний блок виступає для двох типів продуктів: поширення та реклама продуктів замовника та поширення і реклама внутрішніх продуктів компанії. Також у даному блоці може бути розглянуто поширення та реклама послуг самого підприємства. В даному випадку можна виділити наступні ціннісні дії: оцінка загального досвіду підприємства в області маркетингу, кваліфікація працівників відділу, оцінка цільової аудиторії для внутрішніх продуктів компанії, обсяг реклами

на ті чи інші послуги підприємства та інше [3, 71, 74]. Ціннісні дії маркетингу наведені у таблиці 2.4.

Таблиця 2.4 – Маркетинг та продажі

| Ціннісні дії | Оцінка 1, 2, 3, 4, 5 |
|---|---|
| Оцініть знання і досвід фірми в області маркетингу | 3 |
| Кваліфікація працівників відділів маркетингу і збуту | 4 |
| Проводяться маркетингові дослідження, їх якість, системність, як використовуються результати. | 4 |
| Наскільки Ви знаєте свого покупця (віковий склад, рівень доходів тощо) | 4 |
| Наскільки покупці інформовані про переваги продукції підприємства | 4 |
| Обсяг реклами нижче, однаковий або вище ніж у конкурентів | 2 |
| Ви знаєте і відстежуєте зміни в своїй ринковій сфері | 3 |
| Оцініть ефективність каналів розподілу продукції | 3 |
| Ціни на продукцію нижче, однакові або вище ніж у конкурентів | 4 |
| Клієнтам надається знижка у вигляді купонів або в певні дні | 5 |
| Чи існує у фірми програма маркетингу (так, ні, не явно) | 4 |
| Оцініть образ фірми і її продуктів | 4 |
| Оцініть популярність і популярність торгової марки (в порівнянні з конкурентами) | 3 |
| Чи проводяться заходи щодо стимулювання збуту (лотереї, розпродажі, продажу в розстрочку) | 3 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 3 |
| **Разом середня оцінка:** | **3,3** |

Блок обслуговування та підтримки продуктів відповідає за ведення

продукту під час експлуатації. Він є не менш важливим у плані фінансів, адже довгострокова підтримка продукту включає довгострокові контракти. У цьому блоці можна виділити такі ціннісні дії, як оцінка термінів після продажної підтримки, кваліфікація працівників підтримки, оцінюється рівень зворотного зв’язку з клієнтом. Ціннісні дії обслуговування наведені у таблиці 2.5.

Таблиця 2.5 – Маркетинг та продажі

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Термін після продажної підтримки в порівнянні з конкурентами | 3 |
| Кваліфікація працівників підтримки | 3 |
| Чи існує зворотний зв'язок зі замовниками | 4 |
| Чи відомо споживачеві про рівень сервісу підтримки | 5 |
| Чи є додаткові типи сервісу підтримки, що відрізняють Вашу фірму від конкурентів | 3 |
| Рівень після продажного обслуговування | 4 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 3 |
| **Разом середня оцінка:** | 3,6 |

### 2.2.2 Допоміжна діяльність підприємства

Розвиток технології розробки являється допоміжним до блоку виробництва, тому оцінка розвитку технологій напряму впливає на рівень виробництва. У даному випадку можна виділити такі ціннісні блоки, як патентні технології, технології забезпечення якості, спеціальні комп’ютерні програми для забезпечення продажів та інше. Ціннісні дії розвитку технологій розробки наведені у таблиці 2.6.

Таблиця 2.6 – Розвиток технології розробки

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Технологія забезпечує унікальні якості продукції | 3 |
| Оцініть технологію зберігання фінального продукту | 2 |
| Кількість патентів | 1 |
| Технологія забезпечення якості | 3 |
| Спеціальні комп'ютерні програми для забезпечення продажів | 4 |
| Технологія дослідження ринку | 3 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 3 |
| **Разом середня оцінка:** | 2,7 |

Управління людськими ресурсами. Даний блок теж має значний вплив на виробництво, адже саме правильне розподілення трудових ресурсів сприяє оптимізації розробки продукту. Тут можна виділити наступні ціннісні дії: загальна кваліфікація персоналу, програми перенавчання, стабільність у політиці управління персоналом, проводиться оцінка плинності кадрів та проводиться порівняння кваліфікації програмістів з конкурентами. Ціннісні дії управління людськими ресурсами наведені у таблиці 2.7.

Таблиця 2.7 – Управління людськими ресурсами

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Кваліфікація персоналу | 4 |
| Існують програми перенавчання | 3 |
| Стабільна політика в галузі управління персоналом | 3 |
| Плинність кадрів | 4 |
| Кращі програмісти працюють на досліджуваному підприємстві | 2 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 3 |
| **Разом середня оцінка:** | 3,2 |

Інфраструктура підприємства. У даному блоці виділимо наступні цінності дії: оцінка менеджменту компанії в плані зав’язків з оточенням, оцінюються внутрішньо-фірмові зав’язки, проводиться розрахунок фінансової стійкості, оцінюється структура підприємства (таблиця 2.7).

Таблиця 2.8 – Інфраструктура підприємства

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Вищий менеджмент забезпечує ефективні зв'язки з оточенням | 3 |
| Оцініть ефективність внутрішньо-фірмових зв'язків | 4 |
| Імідж фірми | 2 |
| Значення коефіцієнтів фінансової стійкості (відповідають нормі, вище, набагато вище, банкрутство) | 2 |
| Оцініть організаційну структуру фірми | 5 |
| Кваліфікація вищого менеджменту | 3 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 3 |
| Разом середня оцінка: | 3,1 |

Після заповнення осередків загальною таблиці слід зробити висновки по ціннісним блокам, структурувавши їх в окремій таблиці (таблиця 2.9). У лівій частині таблиці розташовуються ціннісні блоки, в правій середні – бали по кожному блоку, вважається загальний середній бал.

Таблиця 2.9 – Загальні висновки по ціннісним блокам

| Ціннісний блок | Середній бал |
|---|---|
| Вхідна логістика | 3,5 |
| Виробництво | 3,4 |
| Вихідна логістика | 4,1 |
| Маркетинг та продажі | 3,3 |
| Обслуговування | 3,6 |

Продовження таблиці 2.9

| Ціннісний блок | Середній бал |
|---|---|
| Загальний середній бал у основній діяльності підприємства | 3,6 |
| Розвиток технології | 2,7 |
| Управління персоналом | 3,2 |
| Інфраструктура | 3,1 |
| Загальний середній бал у допоміжній діяльності підприємства | 3 |
| **Загальна оцінка конкурентоспроможності підприємства:** | 3,3 |

Далі, отримані результати заносяться в рисунок 2.2, що відображає ціннісний ланцюг підприємства.

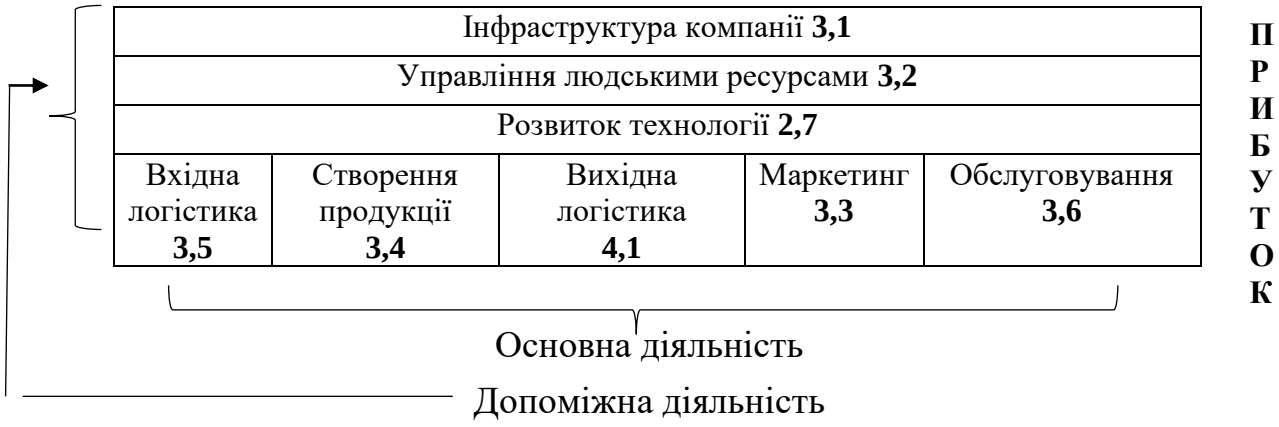


Рисунок 2.2 – Перевірка аналізу конкурентоспроможності підприємства

В результаті, ми отримуємо можливість оцінити реальний стан справ в компанії, визначити точки можливого зростання, блоки неефективного використання капіталу, ціннісні дії, які визначають конкурентну особу компанії і її конкурентні переваги. Графічне представлення ціннісного ланцюга спільно з кількісними експертними оцінками дозволяє недосвідченим в питаннях пов'язаних з конкуренцією українським підприємцям і менеджерам знайти свою власну конкурентну перевагу і створити конкурентоспроможне підприємство.

# 3 РОЗРОБЛЕННЯ МОДЕЛІ ОЦІНКИ ВПЛИВУ ВПРОВАДЖЕННЯ СИСТЕМИ БЕЗПЕРЕРВНОГО РОЗГОРТАННЯ НА ПОКАЗНИКИ КОНКУРЕНТОСПРОМОЖНОСТІ

## 3.1 Впровадження системи безперервного розгортання проектів

Проаналізувавши ринок систем безперервного розгортання було виявлено велику кількість пропозицій інтеграційних систем, як комерційних, так і на базі відкритої ліцензії. Але для точності дослідження було обрано розробку своєї системи безперервного розгортання на основі готового рішення GitLab і Gitlab CI. Розробка системи включає наступні етапи:

1) розгортання системи контролю версій GitLab на власному сервері, початкове налаштування системи для подальшого створення репозиторію проекту;

2) розгортання системи безперервного розгортання GitLab CI на власному сервері, написання власних скриптів обробки репозиторію системою контролю версій, написання скриптів тригерів для автоматичного запуску збірки проекту;

3) створення невеликої демонстраційної програми на основі веб технологій та збереження її у системі контролю версій;

4) створення тестів проекту, для подальшого інтегрування їх до системи безперервної інтеграції;

5) створення коміту-тригеру для початку автоматизованої збірки проекту;

6) аналіз звітів системи безперервної інтеграції, результати збірки та тестування проекту.

Будь-яка команда розробників рано чи пізно стикається з необхідністю використання системи контролю версій. Інакше відстежувати зміни в коді проектів стає складно. Причому чим більше проекти і команди – тим складніше. Сьогодні систем контролю версій існує безліч, одна краща за іншу [51]. Для дослідження обранана система GitLab (рисунок 3.1).

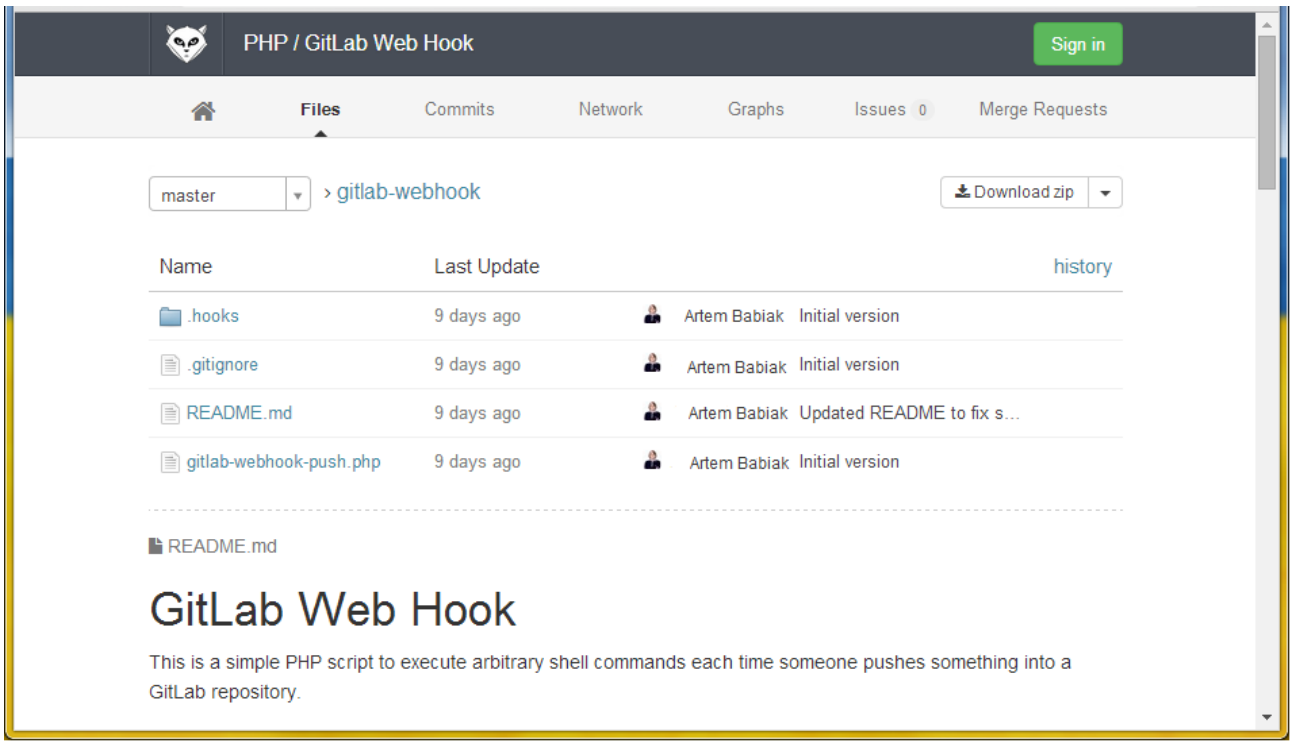


Рисунок 3.1– Приклад проекту у системі контролю версій

GitLab – це веб-додаток для хостингу вихідного коду проектів, заснований на системі контролю версій Git. Своїм функціоналом GitLab дуже нагадує GitHub, проте заточений під командну роботу, в той час як GitHub віддає перевагу індивідуальній роботі.

GitLab існує як у вигляді SAAS веб-сайту з відкритою реєстрацією, так і в якості індивідуального рішення GitLab Community Edition, яке можна встановити на свій сервер і налаштувати під власні потреби. Процес установки досить довгий і вимагає root-доступу до сервера. Для стабільної роботи GitLab вимагає від сервера як мінімум двоядерний процесор і 2 Гб ОЗУ. Така конфігурація забезпечить швидку роботу додатка і підтримку до 500 користувачів. GitLab підтримує безліч різних дистрибутивів Linux, але інструкція по установці розрахована на Debian Ubuntu [56].

Установку та налаштування системи контролю версій можна розбити на кілька етапів:

1) установка необхідних системних утиліт;
2) установка Ruby;
3) створення користувача для SSH-підключень до GitLab;
4) встановлення та налаштування GitLab Shell;

5) встановлення та налаштування бази даних;

6) встановлення та налаштування самого GitLab;

7) встановлення та налаштування Nginx.

Список значний, однак переважна більшість етапів складаються виключно з консольних команд і займають лічені секунди. Тим більше, що якісь з компонентів зі списку вище на більшості серверів вже встановлені і їм потрібно лише незначна конфігурація. Отже, приступимо.

Установка необхідних системних утиліт. Для установки і налаштування всіх компонент, необхідних для роботи GitLab, нам знадобляться: утиліта sudo, набір бібліотек для компіляції Ruby, актуальна версія Git і поштовий сервер.

Перед установкою будь-яких пакетів через утиліту apt-get, слід оновити список джерел та існуючі пакети, виконавши в консолі такі команди (рисунок 3.2):

```
apt-get update -y
apt-get upgrade -y
```

Рисунок 3.2 – Оновлення серверу системи безперервної інтеграції

Тут і далі команди потрібно виконувати від імені користувача root. Першою в списку необхідних компонент йде утиліта sudo. Вона встановлена на більшості експлуатованих серверів. Але якщо раптом ваш сервер зовсім чистий, то для її установки необхідно виконати в консолі наступну команду (рисунок 3.3):

```
apt-get install sudo -y
```

Рисунок 3.3 – Інсталяція програми привілеїв

Тепер встановимо бібліотеки для компіляції Ruby (рисунок 3.4):

```
sudo apt-get install -y build-essential zlib1g-dev libyaml-dev libssl-dev libgdbm-dev libreadline-dev
libncurses5-dev libffi-dev curl openssh-server redis-server checkinstall libxml2-dev libxslt-dev
libcurl4-openssl-dev libicu-dev logrotate
```

Рисунок 3.4 – Інсталяція програмного середовища Ruby

Переконайтеся, що на сервері встановлений git, і його версія не нижче 1.7.10, якщо встановлена застаріла версія, потрібно видалити її (рисунок 3.5):

```
sudo apt-get install -y git-core
git --version
```

Рисунок 3.5 – Інсталяція системи контролю версій

Установка почтового серверу для нормальної роботи системи контролю версій (рисунок 3.6):

```
sudo apt-get install -y postfix
```

Рисунок 3.6 – Інсталяція почтового серверу

Для роботи GitLab потрібно Ruby 2.1.6 і вище. Якщо у вас вже встановлений Ruby нижчої версії, його необхідно видалити перед установкою нової версії (рисунок 3.7):

```
sudo apt-get remove ruby1.8
```

Рисунок 3.7– Видалення старої версії оточення Ruby

Тепер завантажуємо і скомпілюємо Ruby (рисунок 3.8):

```
mkdir /tmp/ruby && cd /tmp/ruby
curl -L --progress http://cache.ruby-lang.org/pub/ruby/2.1/ruby-2.1.6.tar.gz | tar
xz
cd ruby-2.1.6
./configure --disable-install-rdoc
make
sudo make install
```

Рисунок 3.8 – Встановлення нової версії оточення Ruby

Крім самого Ruby нам знадобиться бібліотека bundler (рисунок 3.9):

```
sudo gem install bundler --no-ri --no-rdoc
```

Рисунок 3.9 – Встановлення допоміжної бібліотеки бандлер

Створимо для SSH-підключень користувача git (рисунок 3.10):

```
sudo adduser --disabled-login --gecos 'GitLab' git
```

Рисунок 3.10 – Створення SSH-підключень користувача git

Установка та налаштування GitLab Shell. GitLab Shell – це окрема утиліта для управління SSH-доступом і репозиторіями. Необхідно виконати наступні команди (рисунок 3.11):

```
# Переходим в домашнюю директорию
cd /home/git

# Загружаем исходный код
sudo -u git -H git clone https://gitlab.com/gitlab-org/gitlab-shell.git -b
v1.9.3
cd gitlab-shell

# Создаем конфигурационный файл
sudo -u git -H cp config.yml.example config.yml
```

Рисунок 3.11– Установка та налаштування GitLab Shell

Після виконання команд потрібно відредагувати файл config.yml. У ньому в налаштуванні gitlab_url потрібно вказати майбутню адресу GitLab, наприклад (рисунок 3.12):

```
gitlab_url: http://gitlab.example.com/"
```

Рисунок 3.12 – Редагування конфігурації системи контролю версій

Тепер встановлюємо та налаштовуємо утиліту (рисунок 3.13):

```
sudo -u git -H ./bin/install
```

Рисунок 3.13 – Установка та налаштування GitLab Shell

Установка та налаштування бази даних. Для роботи GitLab вимагає базу даних. Розробники GitLab рекомендують використовувати PostgreSQL, проте підтримка MySQL також присутня. Для установки PostgreSQL, виконайте (рисунок 3.14):

```
sudo apt-get install -y postgresql-9.1 postgresql-client libpq-dev
```

Рисунок 3.14 – Установка та налаштування бази даних

Для установки MySQL (рисунок 3.15):

```
sudo apt-get install -y mysql-server mysql-client libmysqlclient-dev
sudo mysql_secure_installation
```

Рисунок 3.15 – Установка та налаштування MySQL

Під час установки необхідно буде придумати і ввести пароль root користувача MySQL.

Тепер потрібно створити саму базу даних для роботи GitLab. Для цього потрібно виконати кілька SQL-запитів. Почнемо з PostgreSQL. У командному рядку це робиться так (рисунок 3.16):

```
# Логинимся в PostgreSQL
sudo -u postgres psql -d template1

# Создаем пользователя для GitLab
template1=# CREATE USER git;

# Создаем базу данных для GitLab и даем доступ к ней созданному пользователю
template1=# CREATE DATABASE gitlabhq_production OWNER git;

# Выходим из PostgreSQL
template1=# \q
```

Рисунок 3.16 – Створення бази даних для системи контролю версій

Тепер MySQL (необхідно замість **$password** поставити хороший, міцний пароль) (рисунок 3.17):

```
mysql -u root -p
mysql> CREATE USER 'git'@'localhost' IDENTIFIED BY '$password';
mysql> SET storage_engine=INNODB;
mysql> CREATE DATABASE IF NOT EXISTS `gitlabhq_production` DEFAULT CHARACTER SET `utf8` COLLATE
`utf8_unicode_ci`;
mysql> GRANT SELECT, LOCK TABLES, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER ON
`gitlabhq_production`.* TO 'git'@'localhost';
```

Рисунок 3.17– Зміна паролю для доступу

Установка та налаштування самого GitLab. Цей етап найскладніший, тому будемо виконувати його покроково. Перший крок – завантаження початкових кодів GitLab. Робиться це через git (рисунок 3.18):

```
sudo -u git -H clone https://gitlab.com/gitlab-org/gitlab-ce.git b 16-1-stable gitlab
```

Рисунок 3.18 – Установка та налаштування GitLab

Зверніть увагу на прапор **-b 6-8-stable** у команді. На момент написання даної роботи актуальним є GitLab версії 16.1. Однак рекомендовано перед установкою уточнити цю інформацію на офіційному сайті проекту і замінити в цій команді правильну версію. Крім того, разом з GitLab періодично оновлюється і GitLab Shell, це також потрібно врахувати.

Другий крок – розгортання і налаштування (таблиця 3.1). Сюди входять створення необхідних файлів конфігурації, створення необхідних додатку папок і установка прав доступу на вже існуючі папки (рисунок 3.19):

```
# Переходим в папку GitLab
cd /home/git/gitlab

# Создаем файл конфигурации для GitLab
sudo -u git -H cp config/gitlab.yml.example config/gitlab.yml

# Выставляем владельца и права на папки log/ и tmp/
sudo chown -R git log/
sudo chown -R git tmp/
sudo chmod -R u+rwX log/
sudo chmod -R u+rwX tmp/

# Создаем папку для сателлитов и выставляем на нее права
sudo -u git -H mkdir /home/git/gitlab-satellites
sudo chmod u+rwx,g+rx,o-rwx /home/git/gitlab-satellites

# Выставляем права на папки tmp/pids/ и tmp/sockets/
sudo chmod -R u+rwX tmp/pids/
sudo chmod -R u+rwX tmp/sockets/

# Выставляем праав на папку public/uploads/
sudo chmod -R u+rwX  public/uploads

# Создаем файл конфигурации для сервера Unicorn
sudo -u git -H cp config/unicorn.rb.example config/unicorn.rb

# Создаем файл конфигурации для Rack attack
sudo -u git -H cp config/initializers/rack_attack.rb.example config/initializers/rack_attack.rb
```

Рисунок 3.19 – Розгортання та налаштування системи контролю версій GitLab

Таблиця 3.1– Параметри налаштування системи контролю версій

| Налаштування | Розділ | Опис | Приклад |
|---|---|---|---|
| host | gitlab | Сюди треба вписати адресу, за якою буде доступний GitLab | gitlab.example.com |
| email_from | gitlab | Тут потрібно прописати email, з якого буде відправлятися пошта | gitlab@example.com |
| support_emal | gitlab | Адреса тех. підтримки. Якщо його за коментувати, буде використаний адресу з email_from | support@example.com |
| bin_path | git | Встановіть значення / usr / local / bin / git, якщо компілювали git вручну на першому етапі установки | /usr/local/bin/git |

E-mail адресу з email_from також потрібно встановити в конфігурації git (рисунок 3.20):

```
sudo -u git -H git config --global user.name "GitLab"
sudo -u git -H git config --global user.email
"gitlab@example.com"
sudo -u git -H git config --global core.autocrlf input
```

Рисунок 3.20 – Конфігурація системи контролю версій GitLab

Тепер потрібно конфігурувати базу даних. Для PostgreSQL (рисунок 3.21):

```
sudo -u git cp config/database.yml.postgresql config/database.yml
sudo -u git -H chmod o-rwx config/database.yml
```

Рисунок 3.21– Конфігурація системи контролю версій GitLab

Необхідно вказати у файлі config / database.yml логін і пароль для доступу до вашої базі даних. Наступні крок – установка залежностей. Робиться це однієї простої командою. Якщо ви використовуєте PostgreSQL (рисунок 3.22):

```
cd /home/git/gitlab
sudo -u git -H bundle install --deployment --without development test mysql aws
```

Рисунок 3.22 – Встановлення залежностей системи контролю версій GitLab

Тепер потрібно форматувати додаток. Сюди входять ініціалізація бази даних і установка ротації логів (рисунок 3.23):

```
sudo -u git -H bundle exec rake gitlab:setup RAILS_ENV=production
sudo cp lib/support/init.d/gitlab /etc/init.d/gitlab
sudo update-rc.d gitlab defaults 21
sudo cp lib/support/logrotate/gitlab /etc/logrotate.d/gitlab
```

Рисунок 3.23 – Ініціалізація бази даних і установка ротації логів GitLab

Тепер залишилося перевірити статус програми та запустити його. Перевіряється статус такою командою (рисунок 3.24):

```
sudo -u git -H bundle exec rake gitlab:env:info RAILS_ENV=production
```

Рисунок 3.24 – Перевірка статусу програми GitLab

Якщо все добре – додаток можна запускати (рисунок  3.25, 3.26):

```
sudo -u git -H bundle exec rake assets:precompile RAILS_ENV=production
sudo service gitlab start
```

Рисунок 3.25 – Запуск системи контролю версій GitLab

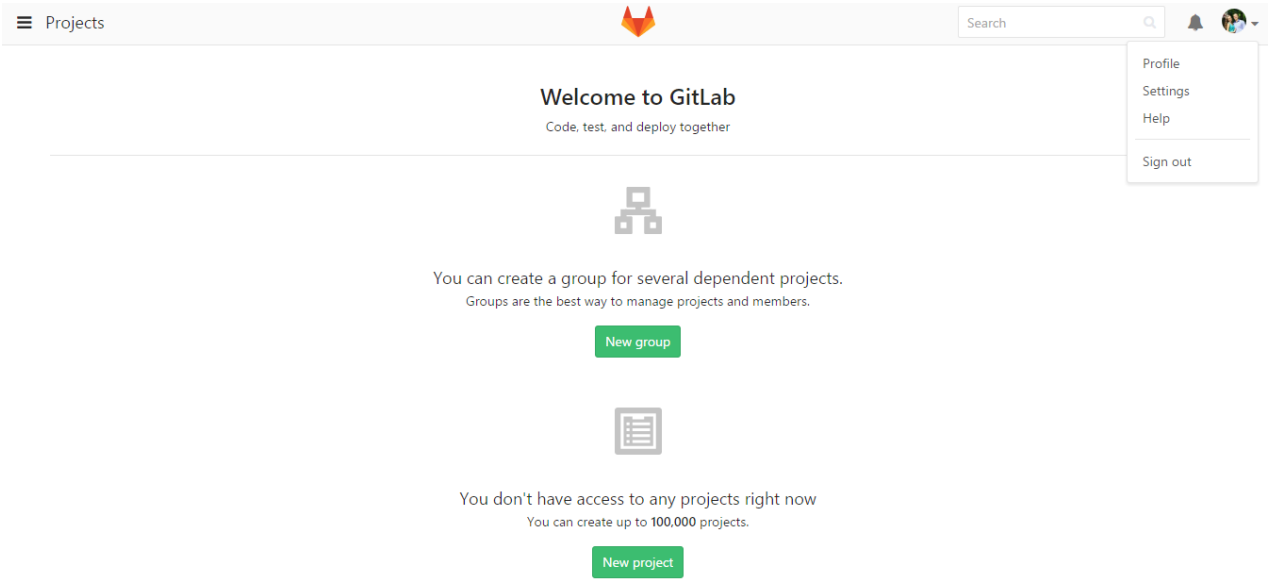


Рисунок 3.26 – Головна панель системи контролю версій GitLab

На цьому установка та налаштування системи контролю версій завершена, далі ми переходимо до налаштування системи безперервного розгортання. Під час роботи над веб-проектами нерідко використовується тестовий сервер. На ньому можна усунути помилки або продемонструвати клієнтам хід роботи і поточний стан проекту. Але для утримання тестового сервера необхідно регулярно розгортати на ньому актуальні зміни. Часто цей процес, крім поновлення файлів, вимагає додаткових операцій: компіляція less і sass стилів або їх аналогів, склеювання і мініфікація скриптів, оптимізація зображень. Все це можна і потрібно автоматизувати за допомогою інструментів на кшталт grunt, але для його запуску потрібно регулярно виконувати команду запуску в командному рядку сервера.

Для вирішення цієї проблеми у таких сервісів, як GitHub і Bitbucket, існують web hooks. Працюють вони дуже просто: коли хтось оновлює код в

репозиторії, сервіс відправляє на вказаний при налаштуванні хука HTTP-адреса запит з даними про оновлення [56, 62]. Є подібний функціонал і в GitLab (рисунок 3.27).

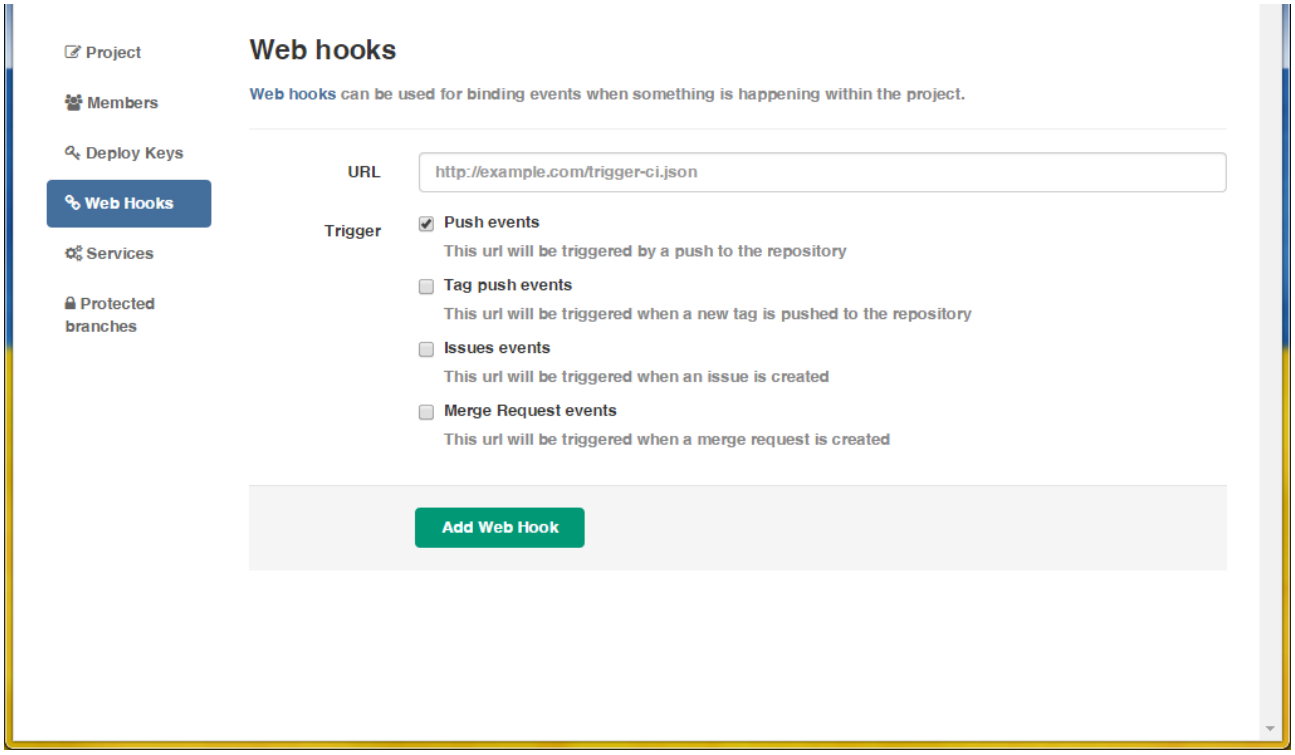


Рисунок 3.27– Веб-хуки системи контролю версій GitLab

Використовуючи GitLab Web Hook, напишемо скрипт, відновлюючий вихідний код проекту. Нам знадобиться можливість виконувати довільні команди в командному рядку, тому сам PHP-скрипт викликатиме shell-скрипт. Подбаймо також і про захист від несанкціонованого запуску нашого скрипта. Для цього сховаємо сам shell-скрипт за межі досяжності веб-сервера, а в PHP-скрипт вбудуємо підтримку паролів. Кожен раз при запуску хука, GitLab передає в нього дані про подію, що викликала активацію хука. У кожної події свій набір даних, але нас цікавить подія push. Серед іншого, серед переданих даних є гілка, яка була оновлена. Використовуючи цю інформацію, додаємо можливість виконання shell-скрипта тільки в разі поновлення конкретної гілки. Код скрипту наведено у дод. А.

Після, того як скрипт буде налаштований, потрібно перейти в розділ Web Hooks налаштувань GitLab і створити новий хук для події push. Процес створення приведений на рисунку 3.28.

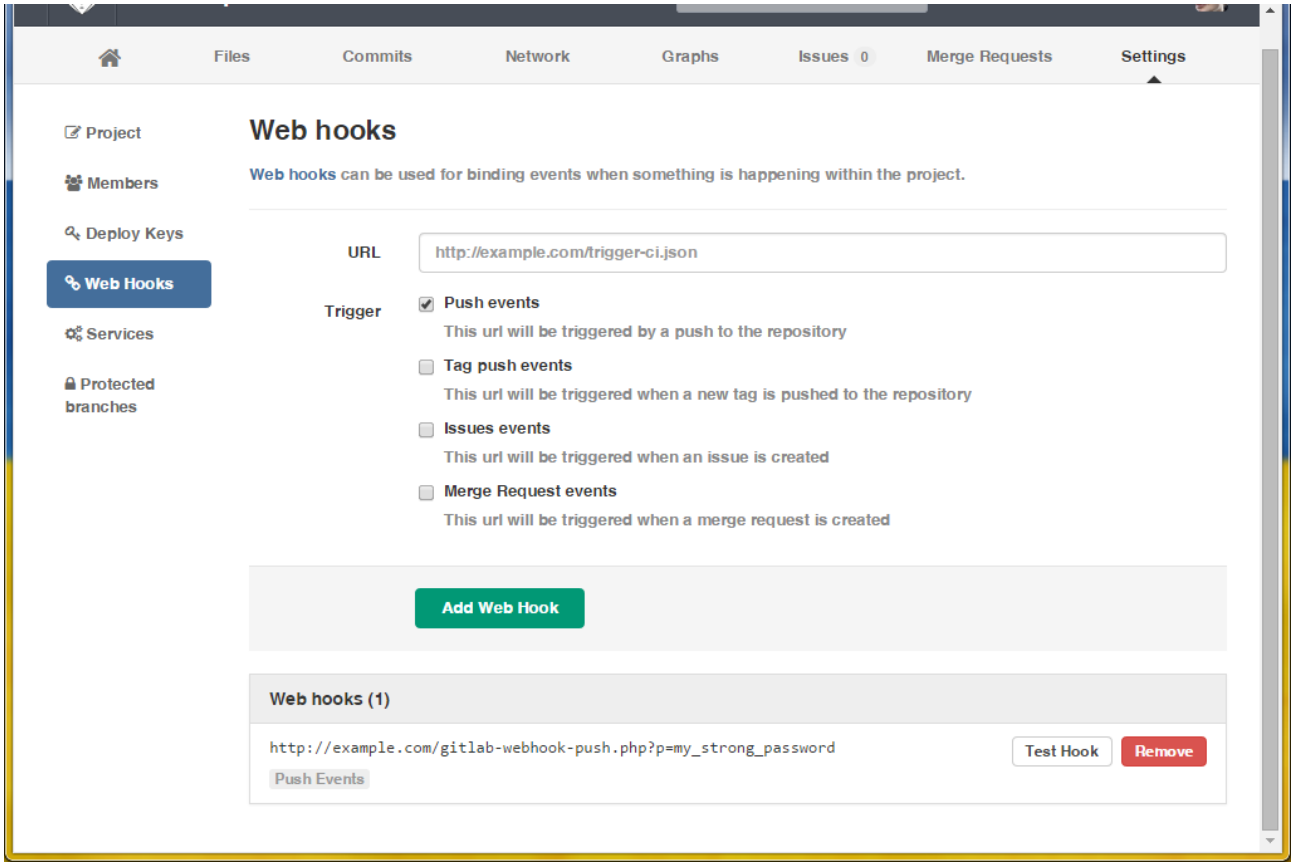


Рисунок 3.28 – Створення веб-хука у системі контролю версій GitLab

Якщо все налаштовано правильно, після наступного оновлення, GitLab відправить на вказану адресу запит, який і запустить наш хук. За замовчуванням, хук виконує команду git pull в корені сайту. Це поведінка може бути змінена налаштування змін в shell-скрипт. Для запуску shell-скрипта у PHP-скрипті використовується PHP-функція exec(), тому переконайтеся що її використання не заборонено конфігурацією сервера.

Використовуючи веб-хуки, можна налагодити чудовий цикл розробки, де найсвіжіші зміни автоматично потрапляють на тестовий сервер, стилі і скрипти автоматично стискаються. Це проміжний крок перед використанням повноцінного CI-сервера (сервера безперервної інтеграції), адже автоматичне розгортання використовує лише технологію веб-хуки.

При установці GitLab в процесі збірки була встановлена і система безперервного розгортання, її використання зводиться до налаштування ранеру, який використовує описані вище веб-хуки. Основною перевагою використання системи безперервної інтеграції є повномірне тестування проекту та створення повноцінних звітів про процес розгортання [48, 73].

## 3.2 Оцінка впливу інтеграції системи безперервного розгортання на процеси розробки проектів

До інтеграції системи безперервного розгортання проектів, підприємство стикалося з безліччю проблем зв’язування незалежних модулів програми, помилки інтеграції доводилося шукати аналізуючи весь код проекту, а тим самим збільшувався термін розробки в цілому. Дані проблеми приносили підприємству великі економічні проблеми, адже замовник продукту не платить за ваші помилки.

Розглянемо приклад оцінки розробки проекту та вплив впровадження системи безперервної інтеграції на терміни розробки. Припустимо що замовних розробляє модульну інформаційну систему для оптимізації документообороту в компанії. Ключовим аспектом даного прикладу є те, що у процесі розробки, систему повинні використовувати саме робітники компанії замовника. На проектом працює декілька команд, що спричиняє безліч помилок інтеграції, але замовник потребує постійного оновлення розроблюваного функціоналу. Така залученість працівників у процес розробки не є рідкістю, адже кінцевим користувачем системи є саме працівники підприємства замовника. Для запобігання розбіжностей функціювання, працівники дають зворотній зв'язок розробникам, що дозволяє повністю зануритися у бізнес процеси підприємства [57, 61].

Наведемо таблицю робіт над проектом, з кількістю використаних трудових ресурсів на ту чи іншу задачу, також приведемо розбіжність до інтеграції системи безперервного розгортання. Роботи над проектом наведені у таблиці 3.2.

Таблиця 3.2 – Задачі на розробку системи документообороту для компанії замовника

| Назва задачі | Спеціалісти | Запланована кількість годин виконання | Витрачена кількість годин | Причина різниці годин |
|---|---|---|---|---|
| Розробка модулю сповіщення персоналу | Відділ розробки модулів сповіщення | 30 годин | 30 годин | Немає |
| Розробка модулю аналізу виконання задач | Відділ розробки модулів аналізу | 50 годин | 55 годин | Неповна технічна інформація |
| Розробка модулю затвердження документів | Відділ розробки модулів затвердження | 20 годин | 15 годин | Немає |
| Розробку модулю сканування документів | Відділ розробки модулів аналізу | 45 годин | 50 годин | Проблеми завантаження драйверів |
| Розробка модулю управління персоналом | Відділ розробки модулів управління персоналом | 80 годин | 100 годин | Неправильна початкова оцінка годин |
| З’єднання модулів в єдину систему | Відділ інтеграції програмних модулів | Невизначено | Невизначено | Причина несумісності кодової бази модулів |

Отже аналізуючи попередню таблицю, ми можемо зробити висновки, що клієнт не отримає свій продукт своєчасно, адже щоб знайти помилку несумісності програмних модулів, потрібно аналізувати усі модулі проекту, що призводить до невизначеності затрат трудових ресурсів, а також створює фінансову проблему підприємству, адже клієнт не платить за помилки компанії розробника.

Здається дана ситуація нерозв’язна, адже розробники не знають де саме створена несумісність програмних модулів. Але давайте розглянемо цю ж саму ситуацію, але з використанням системи безперервного розгортання. Задачі на розробку наведені у таблиці 3.3.

Таблиця 3.3 – Задачі на розробку системи документообороту для компанії замовника

| Назва задачі | Спеціалісти | Запланована кількість годин виконання | Витрачена кількість годин | Причина різниці годин |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| Розробка модулю сповіщення персоналу | Відділ розробки модулів сповіщення | 30 годин | 30 годин | Виявлена проблема інтеграції програмних компонентів |
| Розробка модулю аналізу виконання задач | Відділ розробки модулів аналізу | 50 годин | 55 годин | Неповна технічна інформація |
| Розробка модулю затвердження документів | Відділ розробки модулів затвердження | 20 годин | 15 годин | Немає |

Продовження таблиці 3.3

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Розробку модулю сканування документів | Відділ розробки модулів аналізу | 45 годин | 50 годин | Проблеми завантаження драйверів |
| Розробка модулю управління персоналом | Відділ розробки модулів управління персоналом | 80 годин | 100 годин | Неправильна початкова оцінка годин |
| З’єднання модулів в єдину систему | Непотрібно | Непотрібно | Непотрібно | Непотрібно |

В даному випадку, відділ з’єднання модулів в єдину систему, просто не потрібен, весь проект тестується та розгортається автоматично, що дає клієнту робочу систему, яку він може тестувати власноруч із своїм персоналом. Компанія розробник повідомляє клієнту про новий функціонал, а той в режимі реального часу може починати працювати із системою.

Головна ідея системи безперервного розгортання – це повна автоматизація розгортання та тестування продукту. Уявімо ситуацію коли один модуль програми залежний від іншого. При створенні системи, розробникам потрібно чекати оновлення продукту для початку своєї задачі, адже інтеграція виконується в ручному режимі, система безперервної інтеграції уникає даної проблеми. Тепер продовження розробки залежить лише від швидкості виконання попереднього модулю іншим відділом, адже безперервна інтеграція сама оновить систему після виконання події тригеру, в даному випадку, завершення залежного модулю. Розробникам не потрібно чекати оновлення змін, що значно зменшує термін розробки [61, 65].

## 3.3 Оцінка впливу системи безперервного розгортання на показники конкурентоспроможності підприємства

Повернемось до попередньої оцінки конкурентоспроможності підприємства. В нашому прикладі ми виділяємо блок розробки, адже показники даного блоку значно нижчі, ніж у конкурентів. Зменшенню цього показника сприяла проблема виконання проектів у визначений термін. Проведемо оцінку знову, використовуючи створений метод оцінки, але перерахування виконується лише у блоці виробництва та суміжними з ним допоміжних блоків. Ціннісні дії виробництва наведені у таблиці 3.4, 3.5.

Таблиця 3.4 – Виробництво

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| 1 | 2 |
| Устаткування відповідає сучасним вимогам (гірше, на тому ж рівні або краще, ніж у конкурентів) | 5 |
| Технологія розробки програмного забезпечення, (відповідає вимогам часу, передова, відстала) | 5 |
| Рівень кваліфікації працюючих | 5 |
| Оцініть аналіз розміру обсягу майбутньої розробки | 4 |
| Оцініть кількість працюючих | 3 |
| Яка якість продуктів, що випускаються (бажано порівнювати з продукцією конкурентів) | 5 |
| Чи відповідають цілі керівників і цілі вашого підприємства (не відповідають, відповідають, відповідають частково) | 3 |
| Оцініть виробничі нововведення у Вашому підприємстві (їх кількість, якість, обсяг інвестицій) | 4 |
| Чи є план розвитку виробництва (немає, є, як виконується) | 3 |

Продовження таблиці 3.4

| 1 | 2 |
|---|---|
| Чи є програма підвищення якості продукції | 4 |
| Ваше виробництво здатне перебудовуватися на інші технології? | 4 |
| Наскільки упаковка сприяє продажам, привертає увагу? | 3 |
| Оцініть дизайн продукції, наскільки він відповідає сучасним смакам споживачів | 3 |
| Розмір виробничих витрат (оптимальні, можуть бути знижені, високі) | 4 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 4 |
| **Разом середня оцінка:** | 4,2 |

Таблиця 3.5 – Розвиток технології розробки

| Ціннісні дії | Оцінка 1, 2, 3,4,5 |
|---|---|
| Технологія забезпечує унікальні якості продукції | 4 |
| Оцініть технологію зберігання фінального продукту | 4 |
| Кількість патентів | 1 |
| Технологія забезпечення якості | 5 |
| Спеціальні комп'ютерні програми для забезпечення продажів | 5 |
| Технологія дослідження ринку | 3 |
| Ваш власний показник конкурентоспроможності (якщо такий є) | 4 |
| **Разом середня оцінка:** | 3,7 |

Після змінення осередків загальною таблиці слід зробити висновки по ціннісним блокам, структурувавши їх в окремій таблиці. У лівій частині таблиці розташовуються ціннісні блоки, в правій середні – бали по кожному блоку, вважається загальний середній бал (таблиця 3.6).

Таблиця 3.6 – Загальні висновки по ціннісним блокам

| Ціннісний блок | Середній бал |
|---|---|
| Вхідна логістика | 3,5 |
| Виробництво | 4,2 |
| Вихідна логістика | 4,1 |
| Маркетинг та продажі | 3,3 |
| Обслуговування | 3,6 |
| Загальний середній бал у основній діяльності підприємства | 3,6 |
| Розвиток технології | 3,7 |
| Управління персоналом | 3,2 |
| Інфраструктура | 3,1 |
| Загальний середній бал у допоміжній діяльності підприємства | 3 |
| **Загальна оцінка конкурентоспроможності підприємства:** | 3,53 |

Далі, отримані результати заносяться в рисунок 3.29, що відображає ціннісний ланцюг підприємства:


Рисунок 3.29 – Приклад аналізу конкурентоспроможності підприємства

Як ми можемо бачити показники змінилися, інтеграція системи безперервного розгортання сприяє підвищенню конкурентоспроможності ІТ

компанії. З ростом показників, компанія виходить на новий рівень виробництва, що сприяє підвищенню економічної ефективності.

То яку користь можна отримати про впровадження безперервної інтеграції в своєму проекті? В першу чергу це безболісна інтеграція всього проекту. Інтеграція різних модулів і правок різних програмістів перестає бути проблемою в принципі, вона відбувається "сама" без участі людей і якщо щось не так, ви про це дізнаєтеся. Звичайно, зараз рідкість, що проект має особливу стадію інтеграції, коли з купи різних модулів намагаються зробити додаток, але все ж не треба недооцінювати користь від безперервної інтеграції.

Система не залежить від окремого ПК. Якщо щось не працює на головному сервері – значить воно не працює взагалі. Аргументи програміста, що у нього все працює в даному випадку не допоможуть. Сервер інтеграції стає суддею в таких питаннях і цей суддя неупереджений.

Всі аналізатори коду і тести, які ви використовуєте і написали, обов'язково запускаються над кожною збіркою. Якщо в систему контролю версій потрапив "поганий" код – ви про це дізнаєтеся. І не важливо, чи порушено один із стандартів кодування, або статичний аналізатор коду показує, що в код потрапила потенційна помилка або тести не пройшли, а може просто покриття коду модульними тестами впало нижче необхідного мінімуму. Ви про це дізнаєтеся і зможете вжити заходів.

Більш того, запуск всіх цих аналізаторів корисний не тільки для визначення стану в поточний момент часу, але і для аналізу тенденцій. Можна побачити, коли ваш код став сильно більше, складніше, в яких модулях ця складність сконцентрована. Так, це вимагає наявності та використання відповідного інструментарію.

Чим більше і серйозніше проведена робота з налаштування сервера інтеграції, тим більше користі можна отримати. Якщо ваш сервер просто збирає проект після кожної зміни в коді, то користь від нього не така велика, але і зусиль на нього майже не витрачено.

В цілому, найбільшою перевагою безперервної інтеграції вважається зниження рівня ризиків. Повертаючись назад до попереднього проекту

розробники були вже на фініші (як вважали) довгострокового проекту, все ще не знали скільки ж він буде ще тривати.

В результаті, проекти з безперервною інтеграцією породжують значно менше помилок, як при експлуатації, так і при розробці. Однак варто попередити, що ступінь цієї переваги прямо залежить від того, наскільки хороший ваш набір тестів. Ви зрозумієте, що не дуже складно скласти набір тестів, щоб досягти істотних результатів. Втім, зазвичай проходить деякий час, до того, як команда реально добирається до настільки низької кількості помилок, якого потенційно може добитися. Це досягається постійною розробкою нових і поліпшенням старих тестів [68, 75].

Безперервна інтеграція усуває один з найбільших бар'єрів до частого випуску релізів. Це важливо, тому що дозволяє користувачам відразу починати користуватися новими функціями, розробникам швидко отримувати відгуки на ці функції і, в цілому, процес розробки стає більш інтерактивним. Це дозволяє розбити стіну між замовниками і розробниками, яка є найбільшою перешкодою до успішної розробки програмного забезпечення.

# ВИСНОВКИ

У даній кваліфікаційної роботі виділена важлива проблема оптимізації процесів розробки комерційного програмного забезпечення та вплив впровадження систем безперервної інтеграції на показники конкурентоспроможності. У теперішній час сфера розробки програмного забезпечення перенасичена компаніями, які надають різний спектр послуг, тому що б мати ефективний бізнес потрібно постійно виділятися серед інших компаній, саме показники конкурентоспроможності, які можуть бути підвищені шляхом автоматизації процесів розробки, допомагають якісно оцінити рівень підприємства у сфері комерційного програмного забезпечення. У першу чергу якість роботи та її професіоналізм визначається у внутрішній оптимізації роботи компанії, на сам перед це рівень автоматизації рутинних задач, що передбачає утворення помилок, спричинених людським фактором. Саме впровадження автоматизованої системи безперервної інтеграції кодової бази проекту призводить до зменшення часу на розробку, а також оптимізує процес створення висококваліфікованих систем, що забезпечує надійність майбутнього продукту шляхом постійного тестування.

Аналіз напрямів підвищення конкурентоспроможності дозволив зробити висновок, що існує три основні підходи до розуміння конкурентоспроможності. У першому випадку конкурентоспроможність визначається як змагальність на ринку. У другому випадку конкурентоспроможність розглядається як частина ринкового механізму, яка дозволяє збалансувати попит і пропозицію. У третьому випадку конкурентоспроможність визначається як характеристика, що дозволяє аналізувати тип галузевого ринку. З зазначеними підходами слід погодитися, а використання кожного підходу в кожному конкретному випадку визначається виходячи з управлінських завдань (стратегій) компанії.

У процесі проведення аналізу, було виявлено чіткий вплив оптимізації

процесу розробки на показники конкурентоспроможності, адже після інтеграції системи безперервного розгортання, був значно скорочений час розробки комерційного програмного забезпечення, що призвело до скорочення загального терміну на виконання робіт над майбутнім продуктом замовника. Даний аналіз показує, що при зменшенні часу на розробку проектів, з’являться додатковий пункт впливу на конкуренцію, адже компанія пропонує замовнику випуск продукту у термін, менший ніж у конкурентів, при чому зменшення терміну розробки ніяк не впливає на якість майбутнього програмного забезпечення.

Науково-дослідницьким результатом даної роботи виступає методика оцінки конкурентоспроможності, яка включає у свої розрахунки вплив оптимізації процесів компанії.

Отримані результати можуть бути використані як ІТ компаніями для розуміння важливості автоматизації розробки програмного забезпечення, так і студентами як вектор розвитку майбутнього бізнесу у сфері інформаційних технологій, що сприятиме росту сфери розробки комерційного програмного забезпечення та створить уявлення про важливість і вплив на показники конкурентоспроможності.

Здійсниний аналіз розроблених на сьогодні методів оцінювання рівня конкурентоспроможності підприємства показує, що не існує ідеальної з усіх боків методики комплексного оцінювання конкурентоспроможності підприємства. Виділені недоліки наявних підходів до оцінювання конкурентоспроможності підприємств обумовлюють сильно обмежені можливості практичного застосування більшої їх частини. Наприклад, від методу, за допомогою якого, здійснюється оцінка конкурентоспроможності підприємства невиробничої сфери, суттєво залежить надійність отриманих результатів, простота їх ідентифікації та можливості подальшого застосування.

Для коректної оцінки і подальшого підвищення конкурентоспроможності підприємства розроблено безліч методів, які

можуть застосовуватися як окремо, так і в комплексі, в залежності від завдань, поставлених перед початком проведення оцінки. Різноманіття існуючих сьогодні методів дає можливість підібрати найбільш ефективний і простий метод оцінювання для кожного конкретного підприємства, але у нашому випадку ми спробуємо створити новий метод оцінки, базуючись на вже існуючих.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Авраменко, А. С. Тестування програмного забезпечення [Текст] : навчальний посібник / А. С. Авраменко, В. С. Авраменко, Г. В. Косенюк.– Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284с.

2 Арбон, Д. Як тестують Гугл [Текст] : підручник / Д. Арбон, Д. Каролло, Д. Уіттакер. – 2019. – 303 с.

3 Балабанова, Л. В. Маркетинг [Текст] : підручник / Л. В. Балабанова. – 2-ге вид., перероб. і доп. – К. : Знання-прес, 2006. – 640 с.

4 Бородкіна, Ірина Інженерія програмного забезпечення [Текст] : посібник для студентів вищих навчальних закладів / Ірина Бородкина, Георгій Бородкін. – Київ : ТОВ «Видавництво "Центр навчальної літератури"», 2018. – 204 с.

5 Бондаренко, Е. HR-аналітика в українських компаніях [Електронний ресурс] // HR ЛІГА співтовариство кадровиків та фахівців з управління персоналом – Режим доступу: https://hrliga.com/index.php?module=profession&op=view&id=1844

6 Блек, Р. Ключові процеси тестування [Текст] / Рекс Блек., 2020. – 112 с.

7 Батенко, Л. П. Управління проєктами «Принципи, запропоновані авторами» [Текст] / Л. П. Батенко, О. А. Загородніх, В. В. Ліщинська. – К., 2020. – 231с.

8 Вимоги до якості програмних систем. [Текст] : матеріали одинадцятої міжнародної науково-технічної конференції., 16 – 17 листопада 2023 року. – Баку – Харків – Бельсько-Бяла. – 52 с.

9 Гороховатський, В. О. Методи інтелектуального аналізу та оброблення даних [Текст] : навч. посібник / В. О. Гороховатський, І. С. Творошенко. – Харків : ХНУРЕ, 2021. – 92 с.

10 Говер, І .І. Управління ризиками на проєкті [Текст] : довідковий посібник / І. І. Говер, В. Д. Шапіро – М: Вища школа, 2021.

11 Грей, К. Управління проектами [Текст] : пер. з англ. / К Грей, Е Ларсен М. // «Справа та Сервіс». – 2020. – 528 с.

12 Дегтярьова, Л. М. «Технології розробки програмного забезпечення» [Текст] : навч. посібник для студентів спеціальності 123 «Комп'ютерна інженерія»/ Л. М. Дегтярьова, П. М. Гроза, С. И. Сомов. – Полтава: ПолтНТУ, 2017. – 218 с.

13 Должанський, І. З. Конкурентоспроможність підприємства [Текст] : навч. посібник / І. З. Должанський. – К. : Центр навчальної літератури, 2008. – 384 с.

14 Засоби планування та реалізації ІТ-проектів: рекомендації до вивчення дисципліни [Електронний ресурс] : навч. посіб. для студ. спеціальності 122«Комп'ютерні науки та інформаційні технології», спеціалізації «Інформаційні технології в біології та медицині» / В. С. Якимчук, О. К. Носовець ; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл, 4.64 МВ). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 52 с.

15 Зайцев, Є. О. Основи програмної інженерії [Текст] : навчальний посібник / Є. О. Зайцев. – К.: КНТЕУ, 2017. – 423 с.

16 Зачко, О. Б. Управління проектами: теорія, практика, інформаційні технології [Текст] / О. Б. Зачко, А. І. Івануса, Д. С. Кобилкін. – Львів: ЛДУ БЖД, 2019. – 173 с.

17 До організації виконання та захисту кваліфікаційних робіт другого (магістерського) рівня вищої освіти [Електронне видання] : методичні вказівки для студентів спеціальності 123 «Комп'ютерна інженерія», освітніх програм «Комп'ютерні системи та мережі» та «Системне програмування» усіх форм навчання – Х. : ХНУРЕ, 2021. – 36 с.

18 Керівництво до Зводу знань з управління проектами [Текст] : посібник PMBOK. – 3-те вид. – Американський національний стандарт ANSI/PMI 99-001-2019

19 . Кобилянський, Л. С. Управління ризиками [Текст] : навч. посібник / Л. С. Кобилянський. – К. :МАУП, 2022. – 200с.

20 Майерс, Г. Мистецтво тестування програм [Текст] / Г. Майерс, К. Сандлер, Т. – Баджетт., 2019. – 272 с.

21 Метрики коду програмного забезпечення. Метрика програмного забезпечення [Електронний ресурс] – Режим доступу : https://beasthackerz.ru/uk/programmy/metriki-koda-programmnogo-obespecheniya-metrika-programmnogo-obespecheniya.html

22 Моделі життєвого циклу, принципи і методології розробки програмного забезпечення [Електронний ресурс] – Режим доступу : https://evergreens.com.ua/ua/articles/software-development-metodologies.html

23 Безперервна інтеграція у процесі гнучкої розробки [Електронний ресурс] – Режим доступу : www.ibm.com/developerworks/ru/library/r-continuous-integration-agile-development/

24 Організація управління. (Методологія та філософія діяльності з організації управління) [Текст] : Т.5 курс лекцій / з архіву Г.П. Щедровицького. М., 2020. – 288 с.

25 Основи професійних знань. Національні вимоги до компетенції Спеціалістів [Текст] – М.: Вид-во "Консалтингове Агентство "КУБС Груп - Кооперація, Бізнес-Сервіс", 2021.

26 Палеха, Ю. І. Інформаційний бізнес [Текст] : підручник / Ю. І. Палеха, Ю. І. Горбань. – К.: Вид-во Ліра-К, 2015. – 492 с.

27 Принципи встановлення вимог розробки системи [Електронний ресурс] – Режим доступу : http://org2.knuba.edu.ua/pluginfile.php/28593/mod_resource/content/1

28 Процеси та системи підтримки якості програмних систем [Електронний ресурс] – Режим доступу : https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/16456/1/

29 Розробка Build-Deploy-Test. Безперервна інтеграція [Електронний ресурс] – Режим доступу : habrahabr.ru/company/icl_services/blog/262173/

30 Сервер TeamCity [Електронний ресурс] – Режим доступу : jetbrains.ru/pro-ducts/teamcity/

31 Система керування конфігурацією Ansible [Електронний ресурс] – Режим доступу : blog.selectel.ru/sistema-upravleniya-konfiguraciej-ansible

32 Сидоров, М. О. Вступ до інженерії програмного забезпечення [Електронний ресурс] / М. О. Сидоров – Режим доступу : https://coollib.com/b/222084/read

33 Специфікація вимог до ПЗ [Електронний ресурс] – Режим доступу : https://studfile.net/preview/5375896/

34 Середа, Г. В. Гейміфікація в менеджменті персоналу: зарубіжний та український досвід [Текст] / Г. В. Середа // Економіка і організація управління. – 2017. – Вип. №4. – С. 216–223.

35 Гнилорибов, М. А. Виміри глобального інноваційного розвитку та трансформація промислової політики країн [Текст] / М. А. Гнилорибов // Економіка і організація управління. – 2018. – Вип. 4. – С. 189–197. – Режим доступу: http://nbuv.gov.ua/UJRN/eiou_2018_4_22.

36 Тестування та якість програмного забезпечення [Електронний ресурс] – Режим доступу : https://kpi-fict-ip32.github.io/Blog/s07/software_quality.html

37 Управління проектами [Текст] : підручник / під редакцією В. Д. Шапіро – «Два Три», 2020.

38 Янковий, О. Г. Конкурентоспроможність підприємства: оцінка рівня та напрями підвищення [Текст] : монографія / за заг. ред. О.Г. Янкового. – Одеса : Атлант, 2013. – 470 с.

39 Agile Alliance [Електронний ресурс] – Access mode : http://www.agilealliance.com/

40 Automation Testing Tutorial: What is Automated Testing? [Electronic resource] – Access mode : https://www.guru99.com/automation-testing.html

41 Traits of an Agile Business – Accenture Strategy [Electronic resource] – Access mode : https://www.accenture.com/gb-en/insight-traits-truly-Agile-businesses

42 A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Fifth Edition USA, Project Management Institute – 2013. – 616 p.

43 Agile Software Development [Electronic resource] – Access mode : https://agilesoftwarecourse.org/

44 Automation Testing Tutorial: What is Automated Testing? [Electronic resource] – Access mode : https://www.guru99.com/automation-testing.html

45 Bruegge, Bernd Object-oriented software engineering: using UML, Patterns, and Java [Text] / Bernd Bruegge, Allen Dutoit. – Harlow, UK: Pearson Education Limited, 2014. – 723 p.

46 Bug report [Electronic resource] – Access mode : https://qalight.com.ua/baza-znaniy/bug-report/

47 DevOps Tutorial [Electronic resource] – Access mode : https://www.geeksforgeeks.org/devops-tutorial/

48 DevOps tutorial – an introductijn [Electronic resource] – Access mode : https://azure.microsoft.com/en-ca/solutions/devops/tutorial

49 Isaacs, Trish Agile Teaching and the Agile Manifesto [Text] / Trish Isaacs // Pedagogicon Conference Proceedings. – 2022. – https://encompass.eku.edu/pedagogicon/2021/buckleup/2

50 ISO/IEC TR 19759:2015 Software Engineering – Guide to the software engineering body of knowledge (SWEBOK).

51 Five requirements for deploying an application in a public cloud [Electronic resource] – Access mode : searchcloudcomputing.techtarget.com/ tip/Five-requirements-for-deploying-an-application-in-a-publiccloud.

52 Gaopande, Laxmidhar Software Engineering: A Practical Approach [Text] / Laxmidhar Gaopand. – 2020. – 241 p.

53 GitTutorial [Electronic resource] – Access mode : https://www.w3schools.com/git/

54 Git--distributed-even-if-your-workflow-isnt [Electronic resource] – Access mode : https://git-scm.com/docs/gittutorial

55 20 базових HR – метрик, які допоможуть виміряти ефективність підприємства [Електронний ресурс] – Режим доступу : https://hurma.work/blog/20-bazovih-hr-metrik-yaki-dopomozhut-vimiryati-efektivnist-roboti-kompanii/

56 Learn GitLab with tutorials [Electronic resource] – Access mode : https://docs.gitlab.com/ee/tutorials/

57 Managing the Development of Large Software Systems [Electronic resource] – Access mode : http://facweb.cti.depaul.edu/jhuang/is553/Royce.pdf

58 Oracle Manegemt Human [Text] / Resources та Analyzer [Electronic resource] – Access mode : http://www.oracle.com

59 Pressman, Roger Software Engineering: A Practitioner's Approach [Text] / Roger Pressman, Bruce Maxim. – NY: McGraw-Hill Education, 2019. – 705 p.

60 . «Person Pro 2.0 htp» та «Person Pro 2.0 та SQL» Renaisc [Електронний ресурс] – Режим доступу : Oracle http://personpro.ami.ua

61 Royce, W. Managing the Development of Large Software Systems [Electronic resource] – Access mode : http://facweb.cti.depaul.edu/ jhuang/is553/Royce.pdf.

62 Robbins, J. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics [Text] / J. Robbins – 5th ed. – Sebastopol, CA: O’Reilly Media, 2018. – ISBN-13: 978- 1-491-96020-2.

63 Hayward, Simon The Agile Leader: How to Create an Agile Business in the Digital Age [Text] / Simon Hayward – 1st Edition – KoganPage, 2018. – 224 p. – ISBN-10: 0749482737 | ISBN-13: 978- 0749482732

64 Sommerville, Ian. Software Engineering [Text] / Ian Sommerville. – 10th ed. – Pearson, 2016. – 811 p.

65 Software Engineering Institutehttp [Electronic resource] – Access mode : www.sei.cmu.edu/cmmi/

66 Strategiya razvitiya predpriyatiya [Електронний ресурс] – Режим доступу : https://ukrbukva.net/107494-Strategiya-razvitiya-predpriyatiya.html

67 Sutherland, J. A Scrum book: the spirit of the game [Text] / J. Sutherland, J. O. Coplien. – Pragmatic Bookshelf, 2019. – Access mode : https://www.scrum.org/

68 Scrum.org [Electronic resource] – Access mode : https://www.scrum.org/

69 SW-CMM [Electronic resource] – Access mode : http://www.sei.cmu.edu/cmm

70 SAP HRuman Resources Management System [Electronic resource] – Access mode : http://www.sap.com

71 The Simplest Way to Build and Deploy Web Applications to the Cloud? [Electronic resource] – Access mode : www.ctl.io/blog/post/the-simplest-way-to-build-and-deploy-web-applications-to-the-cloud.

72 Technology revolutionizes our giving methods. Fidelity Charitable. [Electronic resource] – Access mode : https://www.fidelitycharitable.org/insights/2021-future-of-philanthropy/technology.html

73 Voorhees, David Guide to Efficient Software Design An MVC Approach to Concepts, Structures, and Models [Text] / David Voorhees. – Springer Nature Switzerland AG, 2020. – 519 p.

74 What are Web Services? Architecture, Types, Example [Electronic resource] – Access mode : https://www.guru99.com/web-service-architecture.html

75 What is Continuous Integration? The Essentials You Need to Know [Electronic resource] – Access mode : https://www.cloudbees.com/continuous-delivery/continuous-integration