ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Ефективне управління модульними архітектурами у розробці програмного забезпечення для корпоративного простору інжинірингових послуг»

Ступінь вищої освіти – магістр

Спеціальність – 073 «Менеджмент»

Освітня програма «Agile-технології розробки програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Керівники:
д.е.н., доц., професор кафедри ІММС
Ольга ОРЛОВА-КУРИЛОВА

Керівник:
к. фіз-мат. н., доц.,
доцент кафедри ІММС
Іван КРИКУН

Виконав:
здобувач групи МЕН/Agile-23м
Вадим САВЧУК

Київ, 2024 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри інформаційного менеджменту, математики та статистики
_____Денис БАЛДИК
«__»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
САВЧУК ВАДИМ ОЛЕКСАНДРОВИЧ

Тема роботи	Ефективне управління модульними архітектурами у розробці програмного забезпечення для корпоративного простору інжинірингових послуг
Номер та дата наказу про затвердження теми	№ 56-5 від 27 червня 2024 р.
Коротка постановка завдання	Дослідити Agile-методології та їх адаптацію до управління модульними архітектурами. Визначити специфіку корпоративного простору інжинірингових послуг та вимоги до програмного забезпечення. Провести “Case study” аналізу використання модульної архітектури в інжиніринговій компанії.
Посилання на джерела інформації (не більше п’яти найменувань, які рекомендує науковий керівник)	1. Сайт компанії ТОВ “Прогрестех-Україна” URL: https://progresstech.ua/ 2. Сайт компанії LLC “Boeing” URL: https://www.boeing.com/ 3. Сайт Української асоціації IT-професіоналів: https://itukraine.org.ua/
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне та/або практичне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблематику в галузі управління та адміністрування, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів Agile технологій.

Дата видачі завдання «14» липня 2024 р.

Керівник Ольга ОРЛОВА-КУРИЛОВА

Керівник Іван КРИКУН

Здобувач Вадим САВЧУК

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження та керівника	01.07.2024 р.	Виконано
2	Формування теми та призначення керівника	08.07.2024 р.	Виконано
3	Затвердження теми кваліфікаційної роботи	09.07.2024 р.	Виконано
4	Затвердження завдання на кваліфікаційну роботу	15.07.2024 р.	Виконано
Основний етап			
5	Розробка концепції кваліфікаційної роботи	22.07.2024 р.	Виконано
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів.	29.07.2024 р.	Виконано
7	Затвердження розширеної постановки завдання. Підготовка та подання керівнику розділу 1 кваліфікаційної роботи	18.09.2024 р.	Виконано
8	Проєктування інформаційної системи. Підготовка та подання керівнику розділу 2 кваліфікаційноїроботи	18.09.2024 р.	Виконано
9	Реалізація інформаційної системи. Підготовка та подання керівнику розділу 3 кваліфікаційної роботи	25.09.2024 р.	Виконано
10	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	01.10.2024 р.	Виконано
11	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівнику доопрацьованого варіанту кваліфікаційної роботи	04.10.2024 р.	Виконано
Завершальний етап			
12	Представлення рукопису для перевірки на плагіат	07.10.2024 р.	Виконано
13	Підготовка презентації та доповіді на передзахист	07.10.2024 р.	Виконано
14	Передзахист кваліфікаційної роботи	08-11.10.2024 р.	Виконано
15	Технічна самоекспертиза роботи на відповідність вимогам до оформлення та виправлення недоліків	08-11.10.2024 р.	Виконано
16	Експертиза роботи керівником та зовнішнім експертом	14.10.2024 р.	Виконано
17	Доопрацювання доповіді та презентації для захисту	18.10.2024 р.	Виконано
18	Захист кваліфікаційної роботи	21-25.10.2024 р.	Виконано

Керівник Ольга ОРЛОВА-КУРИЛОВА

Керівник Іван КРИКУН

Здобувач Вадим САВЧУК

АНОТАЦІЯ

Савчук В.О. Ефективне управління модульними архітектурами у розробці програмного забезпечення для корпоративного простору інжинірингових послуг.

Пояснювальна записка кваліфікаційної роботи за спеціальністю 073 – Менеджмент (освітня програма – Agile-технології розробки програмного забезпечення), СО Магістр. – ВНЗ «Університет економіки та права «КРОК», Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра інформаційного менеджменту, математики та статистики, Київ, 2024р.

Кваліфікаційна робота присвячена дослідженню ефективного управління модульними архітектурами у розробці програмного забезпечення для корпоративного простору інжинірингових послуг з використанням Agile-методологій. В роботі проаналізовано сутність, типи, переваги та недоліки модульних архітектур, а також досліджено Agile-методології та їх адаптацію до управління такими архітектурами. Визначено специфіку корпоративного простору інжинірингових послуг та вимоги до програмного забезпечення в цьому секторі. Розроблено рекомендації щодо планування, організації, розробки, тестування, впровадження та підтримки модульних систем в інжинірингових компаніях. Проведено “Case study” аналізу використання модульної архітектури на прикладі бізнес-утиліти “WorkHive”, розроблено модель процесу управління модульною архітектурою та оцінено ефективність запропонованого підходу.

Ключові слова: модульні архітектури, Agile-методології, інжинірингові послуги, розробка програмного забезпечення, Scrum, Kanban, “Case study”, моделювання, ефективність.

Табл. 11 Мал. 8 Бібліограф.: 17 найм.

ANNOTATION

Savchuk V.O. Effective management of modular architectures in software development for the corporate space of engineering services.

Project explanatories note by specialty 073 - Management (educational program - Agile software development technologies). – «KROK» University, Educational and Scientific Institute of information and communication technologies, Department of Information Management, Mathematics and Statistics, Kyiv, 2024.

The thesis is devoted to the study of effective management of modular architectures in software development for the corporate space of engineering services using Agile methodologies. The paper analyzes the essence, types, advantages and disadvantages of modular architectures, and also explores Agile methodologies and their adaptation to the management of such architectures. The specifics of the corporate space of engineering services and the requirements for software in this sector are determined. Recommendations for planning, organization, development, testing, implementation and maintenance of modular systems in engineering companies are developed. A “Case study” of the use of modular architecture on the example of the business utility “WorkHive” is carried out, a model of the modular architecture management process is developed, and the effectiveness of the proposed approach is evaluated.

Key words: modular architectures, Agile methodologies, engineering services, software development, Scrum, Kanban, “Case study”, modeling, efficiency.

Tabl. 11. Fig. 8. Bibliography: 17 Items.

ЗМІСТ

ВСТУП

Сучасний корпоративний простір інжинірингових послуг характеризується високою динамікою, зростаючою складністю проєктів та необхідністю швидкої адаптації до змін ринку. У цих умовах розробка програмного забезпечення для інжинірингових компаній потребує нових підходів, які забезпечують гнучкість, масштабованість та ефективне управління. Одним з таких підходів є використання модульних архітектур у поєднанні з Agile-методологіями.

Модульні архітектури дозволяють розробляти програмне забезпечення як набір незалежних модулів, що взаємодіють між собою. Це забезпечує ряд переваг, таких як:

1. Гнучкість та адаптивність:

- модулі можна легко змінювати, додавати або видаляти, що дозволяє швидко реагувати на нові вимоги замовників та зміни ринку;

2. Масштабованість:

- модульні системи легко масштабуються шляхом додавання нових модулів або збільшення потужності існуючих;

3. Повторне використання:

- модулі можна використовувати в різних проєктах, що зменшує час та витрати на розробку;

4. Спрощення розробки та тестування:

- розробка та тестування окремих модулів є простішим, ніж робота з монолітними системами.

Agile-методології, зі своєю ітеративною природою та орієнтацією на спільну роботу, ідеально підходять для управління розробкою модульних систем. Вони дозволяють швидко адаптуватися до змін, ефективно співпрацювати з замовниками та забезпечувати високу якість програмного забезпечення.

Актуальність теми дослідження зумовлена зростаючим попитом на гнучке та адаптивне програмне забезпечення в інжинірингових компаніях, а також необхідністю ефективного управління розробкою модульних систем.

Мета дослідження полягає в розробці рекомендацій щодо ефективного управління модульними архітектурами у розробці програмного забезпечення для корпоративного простору інжинірингових послуг з використанням Agile-методологій.

Для досягнення мети поставлено наступні завдання:

1. Проаналізувати сутність, типи, переваги та недоліки модульних архітектур.

2. Дослідити Agile-методології та їх адаптацію до управління модульними архітектурами.

3. Визначити специфіку корпоративного простору інжинірингових послуг та вимоги до програмного забезпечення.

4. Розробити рекомендації щодо планування, організації, розробки, тестування, впровадження та підтримки модульних систем в інжинірингових компаніях.

5. Провести “Case study” аналізу використання модульної архітектури в інжиніринговій компанії.

6. Розробити модель процесу управління модульною архітектурою в інжиніринговій компанії.

7. Оцінити ефективність запропонованого підходу.

Об'єктом дослідження є процес управління модульними архітектурами в розробці програмного забезпечення.

Предметом дослідження є методи та інструменти ефективного управління модульними архітектурами в Agile-середовищі для корпоративного простору інжинірингових послуг.

Методи дослідження:

- аналіз літератури;
- “Case study”;

- моделювання;
- аналіз документації;
- опитування експертів.

Наукова новизна одержаних результатів полягає в розробці комплексного підходу до управління модульними архітектурами в інжиніринговых компаніях, що враховує специфіку галузі та оснований на Agile-методологіях.

Практичне значення одержаних результатів полягає в можливості їх використання інжиніринговими компаніями для підвищення ефективності розробки програмного забезпечення, скорочення часу виведення продукту на ринок та підвищення його якості.

Розглянемо теоретичну апробацію результатів дослідження на майданчику ТОВ “Прогрестех-Україна”.

Результати дослідження були апробовані на базі ТОВ “Прогрестех-Україна” - провідної інжиніринговой компанії, що спеціалізується на наданні інжиніринговых послуг у сфері авіації та машинобудування, проєктуванні аеропортів та розробці програмного забезпечення для авіаційної галузі.

Компанія має значний досвід у використанні модульних архітектур та Agile-методологій.

В рамках теоретичної апробації було проведено наступне:

1. Презентація результатів дослідження - ключові висновки та рекомендації дослідження були представлені на внутрішньому мітапі для співробітників ТОВ “Прогрестех-Україна”, що займаються розробкою програмного забезпечення;

2. Обговорення з експертами - проведено обговорення результатів дослідження з провідними експертами компанії в галузі розробки програмного забезпечення та управління проєктами. Експерти надали свої відгуки та пропозиції щодо запропонованого підходу до управління модульними архітектурами;

3. Аналіз відповідності - проведено аналіз відповідності запропонованого підходу до існуючих процесів розробки програмного забезпечення в ТОВ “Прогрестех-Україна”;

Моделювання на прикладі бізнес утиліти “WorkHive”: запропонована модель процесу управління модульною архітектурою була змодельована на прикладі бізнес-утиліти “WorkHive”, описаної в “Case study”. Це дозволило продемонструвати практичне застосування розробленого підходу в реальних умовах компанії.

Результати апробації:

- експерти ТОВ “Прогрестех-Україна” позитивно оцінили актуальність та практичну значимість дослідження;
- запропонований підхід до управління модульними архітектурами був визнаний ефективним та перспективним для використання в компанії;
- було отримано ряд цінних пропозицій та рекомендацій щодо подальшого розвитку та вдосконалення запропонованого підходу.

Теоретична апробація результатів дослідження на майданчику ТОВ “Прогрестех-Україна” підтвердила їх актуальність, практичну значимість та ефективність. Отримані відгуки та пропозиції експертів будуть враховані при подальшому вдосконаленні запропонованого підходу.

Обмеження та проблеми які можуть бути виявлені при апробації, що пов'язані зі специфікою компанії:

1. Специфіка інжинірингових послуг - ТОВ “Прогрестех-Україна” спеціалізується на наданні інжинірингових послуг у сфері авіації та машинобудування, проєктуванні аеропортів та розробці програмного забезпечення для авіаційної галузі, що може мати свою специфіку та вимоги до програмного забезпечення (наприклад, підвищені вимоги до безпеки та надійності). Це може обмежити застосовність деяких рекомендацій, розроблених в рамках дослідження;

2. Існуючі процеси та інструменти - в компанії вже можуть бути встановлені процеси розробки програмного забезпечення та

використовуватися певні інструменти. Впровадження нових підходів може потребувати адаптації або навіть змін в існуючих процесах, що може бути складним завданням;

3. Корпоративна культура - корпоративна культура компанії може впливати на сприйняття нових ідей та готовність до змін;

Обмеження, що пов'язані з дослідженням:

1. Обмежений обсяг вибірки: - теоретична апробація проводилася на одному підприємстві, що може обмежувати узагальнення результатів на інші інжинірингові компанії;

2. Суб'єктивність оцінок експертів - оцінки експертів можуть бути суб'єктивними та залежати від їхнього досвіду та поглядів.

Проблеми, які можуть виникнути:

1. Опір змінам - деякі співробітники можуть чинити опір впровадженню нових підходів, особливо якщо це потребує змін в їхній роботі;

2. Нестача ресурсів - впровадження нових підходів може потребувати додаткових ресурсів (час, фінанси, навчання персоналу), яких може не вистачати;

3. Технічні труднощі - можуть виникнути технічні труднощі при інтеграції нових інструментів або модулів в існуючу інфраструктуру.

Щоб мінімізувати вплив цих обмежень та проблем, важливо:

1. Врахувати специфіку - необхідно чітко вказати, що рекомендації розроблені з урахуванням специфіки ТОВ “Прогрестех-Україна”, авіаційної галузі та регуляторних органів;

2. Запропонувати адаптацію - розробка рекомендацій щодо адаптації запропонованого підходу до різних типів інжинірингових компаній;

3. Обґрунтувати ефективність - навести докази ефективності запропонованого підходу, навіть за умови обмежених даних апробації (наприклад, результати моделювання, експертні оцінки).

Враховуючи специфіку ТОВ “Прогрестех-Україна” та виявлені обмеження, пропоную наступні етапи впровадження рекомендацій дослідження:

Етап 1. Підготовка (1-2 місяці):

1. Аналіз існуючих процесів - детальний аналіз існуючих процесів розробки програмного забезпечення в компанії, виявлення сильних та слабких сторін, оцінка готовності до змін;

2. Адаптація рекомендацій - адаптація рекомендацій дослідження до специфіки ТОВ “Прогрестех-Україна”, враховуючи вимоги авіаційної галузі та особливості компанії;

3. Розробка плану впровадження - створення детального плану впровадження з визначенням термінів, ресурсів, відповідальних осіб та ключових показників ефективності;

4. Інформування та навчання - проведення презентацій та навчальних семінарів для співробітників компанії, щоб ознайомити їх з новими підходами та інструментами.

Етап 2. Пілотне впровадження (3-4 місяці):

1. Вибір пілотного проєкту - вибір пілотного проєкту для апробації запропонованого підходу в реальних умовах, пропонується бізнес утиліта “WorkHive”;

2. Формування команди - створення команди з досвідчених спеціалістів, які будуть відповідальні за впровадження нових підходів в пілотному проєкті;

3. Впровадження та моніторинг - поетапне впровадження рекомендацій дослідження в пілотному проєкті з постійним моніторингом та аналізом результатів;

Етап 3. Поширення та масштабування (6-12 місяців):

1. Аналіз результатів пілотного проєкту - аналіз результатів пілотного впровадження, виявлення проблем та успіхів, корекція плану впровадження;

2. Поширення на інші проєкти - поступове поширення запропонованого підходу на інші проєкти компанії з урахуванням отриманого досвіду, за бажання керівництва компанії;

3. Масштабування та оптимізація - масштабування впровадження на всю компанію з одночасною оптимізацією процесів та інструментів;

Етап 4. Безперервне вдосконалення:

1. Збір та аналіз зворотнього зв'язку - регулярний збір та аналіз зворотнього зв'язку від співробітників компанії щодо ефективності запропонованого підходу;

2. Вдосконалення процесів - безперервне вдосконалення процесів розробки програмного забезпечення на основі отриманих даних та аналізу найкращих практик;

Важливі аспекти:

1. Залучення керівництва - активна підтримка та залучення керівництва компанії є ключовим фактором успішного впровадження;

2. Комунікація - ефективна комунікація між учасниками проєкту дозволить уникнути непорозумінь та забезпечити координацію дій;

3. Гнучкість - план впровадження має бути гнучким та адаптуватися до змін в компанії та на ринку.

Впровадження рекомендацій дослідження - це тривалий процес, що потребує часу, ресурсів та зусиль. Але за умови належного планування та виконання він дозволить ТОВ “Прогрестех-Україна” значно підвищити ефективність розробки програмного забезпечення та конкурентоспроможність на ринку.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МОДУЛЬНИХ АРХІТЕКТУР ТА AGILE-МЕТОДОЛОГІЙ В КОНТЕКСТІ ІНЖИНІРИНГОВИХ ПОСЛУГ

В сучасному світі розробка програмного забезпечення для корпоративного сектору, особливо в динамічній сфері інжинірингових послуг, вимагає гнучких та адаптивних рішень. Модульні архітектури, що дозволяють створювати програмне забезпечення як набір взаємодіючих модулів, стали відповіддю на цей виклик. Поєднуючи модульність з Agile-методологіями, розробники отримують потужний інструментарій для створення складних, але в той же час гнучких і масштабованих систем.

Цей розділ присвячений теоретичному обґрунтуванню використання модульних архітектур та Agile-методологій в контексті інжинірингових послуг. В ньому будуть розглянуті наступні аспекти:

1. Сутність модульних архітектур:

- Визначення, основні принципи, типи модульних архітектур (мікросервіси, мікрофронтенди тощо), їх переваги та недоліки;

2. Agile-методології:

- Основні принципи та практики Agile (Scrum, Kanban, XP), їх роль в управлінні модульними архітектурами;

3. Специфіка інжинірингових послуг:

- Особливості розробки програмного забезпечення для інжинірингових компаній, типові вимоги до таких систем (надійність, безпека, масштабованість);

Розуміння цих теоретичних основ є необхідним для ефективного управління розробкою програмного забезпечення в інжинірингових компаніях та досягнення успіху в цьому динамічному середовищі.

1.1 Модульні архітектури в розробці програмного забезпечення: сутність, типи, переваги та недоліки.

В умовах зростаючої складності програмних систем та необхідності швидкої адаптації до змін, модульні архітектури стають все більш популярним підходом до розробки програмного забезпечення.

1.1.1 Сутність модульних архітектур

Модульна архітектура - це підхід до проєктування програмного забезпечення, при якому система розробляється як набір незалежних, взаємозамінних модулів, кожен з яких виконує певну функцію.

Основні принципи модульних архітектур:

1. Інкапсуляція:

- Кожен модуль приховує свою внутрішню реалізацію, надаючи доступ до своєї функціональності через чітко визначений інтерфейс;

2. Слабка зв'язність:

- Модулі мають мінімальну залежність один від одного, що дозволяє змінювати або замінювати їх без впливу на інші модулі;

3. Висока когезія:

- Елементи всередині модуля тісно пов'язані між собою та виконують спільну функцію;

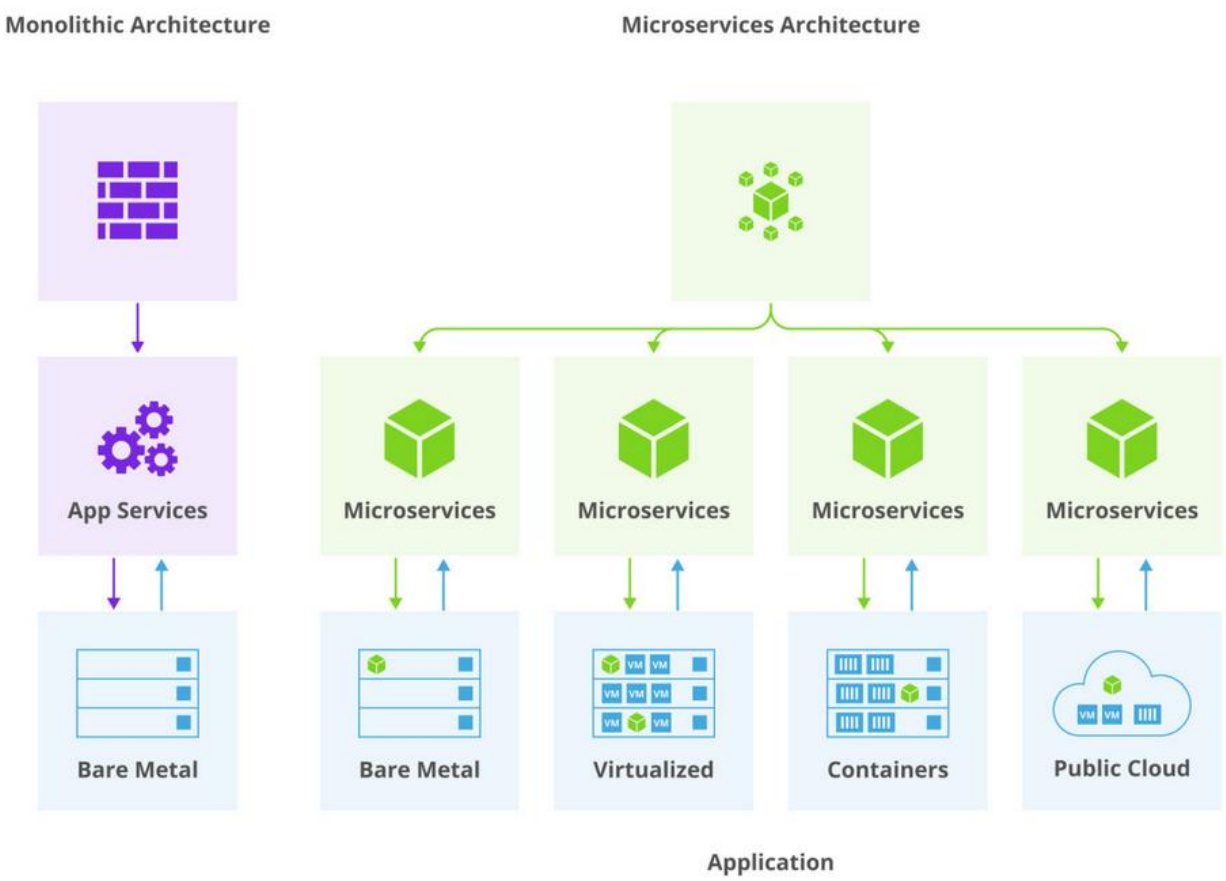
4. Повторне використання:

- Модулі можуть бути використані в різних проєктах, що зменшує час та витрати на розробку;

Типи модульних архітектур:

1. Мікросервіси**:**

- Архітектура, де додаток складається з незалежних сервісів, що взаємодіють між собою через мережу. Для візуалізації використаємо малюнок 1.1;



Малюнок 1.1 Як працює архітектура мікросервісів

Джерело: розроблено blog.colobridge.net

Мікросервісна архітектура:

1. Мікрофронтенди**:**

- Підхід до розробки фронтенду, де інтерфейс користувача складається з незалежних модулів, розроблених різними командами;

2. Модулі на рівні коду:

- Організація коду в модулі, що дозволяє розділити функціональність та спростити підтримку коду (наприклад, використання пакетів в Java або модулів в Python);

Переваги модульних архітектур:

1. Гнучкість та адаптивність:

- Легко змінювати, додавати або видаляти модулі, швидко реагуючи на зміни вимог;

2. Масштабованість:

- Легко масштабувати систему шляхом додавання нових модулів або збільшення потужності існуючих;

3. Повторне використання:

- Зменшення часу та витрат на розробку за рахунок повторного використання модулів;

4. Спрощення розробки та тестування:

- Розробка та тестування окремих модулів є простішим, ніж робота з монолітними системами;

5. Підвищення надійності:

- Збій в одному модулі не впливає на роботу інших модулів;

Недоліки модульних архітектур:

1. Складність управління залежностями:

- Необхідність управляти залежностями між модулями може бути складним завданням;

2. Проблеми з інтеграцією:

- Інтеграція модулів, розроблених різними командами, може бути складною;

3. Підвищені вимоги до інфраструктури:

- Модульні системи можуть вимагати більш складної інфраструктури (наприклад, системи оркестрації для мікросервісів);

Модульні архітектури є ефективним підходом до розробки складних програмних систем, особливо в динамічних середовищах, таких як корпоративний простір інжинірингових послуг. Однак, важливо враховувати їх недоліки та ретельно планувати процес розробки та впровадження.

1.2 Agile-методології: принципи, практики та їх адаптація до управління модульними архітектурами

Agile-методології - це ітеративний та інкрементний підхід до розробки програмного забезпечення, що орієнтований на гнучкість, співпрацю та швидку реакцію на зміни. Вони ідеально підходять для управління модульними архітектурами, де необхідно ефективно координувати розробку незалежних модулів.

Основними принципами Agile є:

1. Люди та взаємодія важливіші за процеси та інструменти.

2. Робочий продукт важливіший за вичерпну робочу документацію.

3. Співпраця із замовником важливіша за договірні зобов'язання.

4. Готовність до змін важливіша за слідування плану.

Ключовими практиками Agile є:

1. Ітеративна розробка:

- Розробка програмного забезпечення в короткі ітерації (спринти), кожна з яких завершується робочим продуктом;

2. Безперервна інтеграція:

- Часта інтеграція коду в загальний репозиторій та автоматизоване тестування;

3. Спільна робота:

- Тісна співпраця між розробниками, тестувальниками та замовником;

4. Зворотній зв'язок:

- Регулярний зворотній зв'язок від замовника та користувачів;

Адаптація Agile до управління модульними архітектурами використовує:

1. Незалежні команди:

- Створення незалежних команд, відповідальних за розробку окремих модулів;

2. Чіткі інтерфейси:

- Визначення чітких інтерфейсів між модулями для забезпечення їх незалежної розробки та інтеграції.

3. Автоматизоване тестування:

- Використання автоматизованого тестування для забезпечення якості та сумісності модулів.

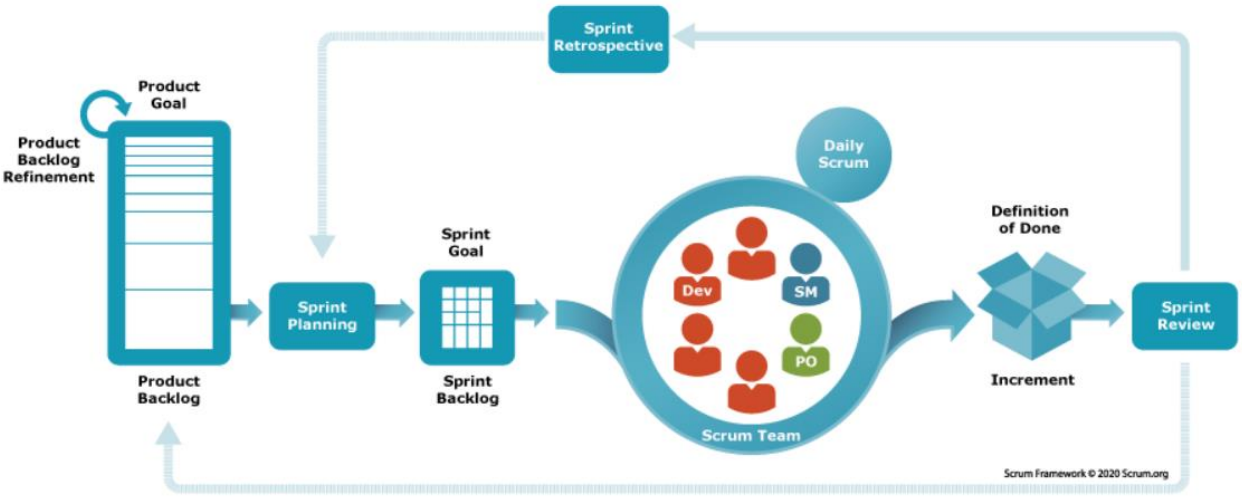
4. Безперервна інтеграція/безперервна поставка (CI/CD):

- Автоматизація процесів збірки, тестування та розгортання модулів;

Розглянемо популярні Agile-методології:

1. Scrum:

- Фреймворк для управління проєктами, що базується на ітераціях (спринтах) та ролях (Product Owner, Scrum Master, розробники) наочно проілюстрований на малюнку 1.1;



Малюнок 1.1 Scrum framework

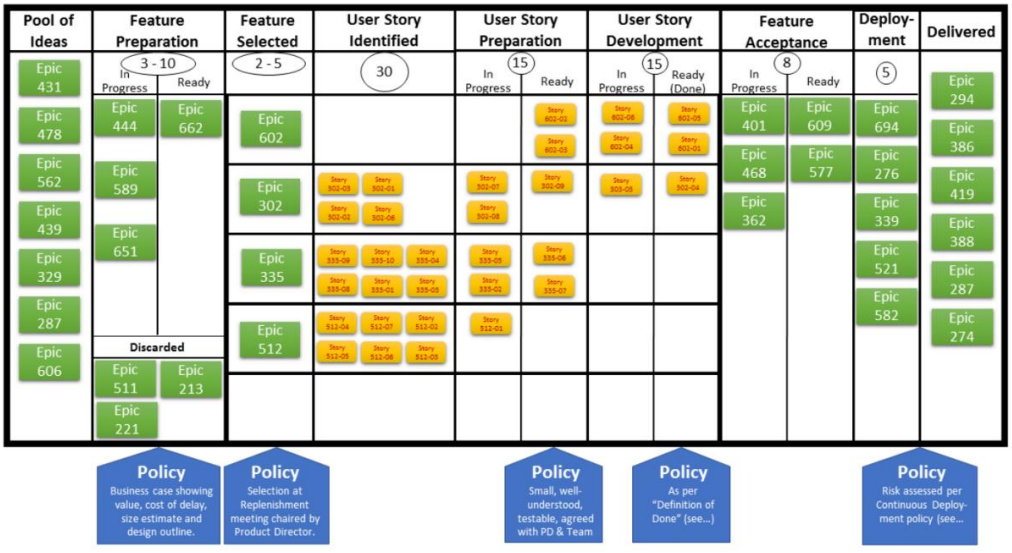
Джерело: розроблено Scrum.org

2. Kanban:

- Метод візуалізації робочого процесу та обмеження кількості завдань в роботі для забезпечення ефективності, елементи якого проілюстровані на малюнках 1.2 та 1.3;

Малюнок 1.2 Kanban board

Джерело: розроблено Wikipedia.org



Малюнок 1.3 Kanban board в розробці програмного забезпечення

Джерело: розроблено Wikipedia.org

3. Extreme Programming (XP):

- Набір практик, спрямованих на підвищення якості коду та задоволення потреб замовника (парне програмування, тестування перед написанням коду тощо).

Agile-методології дозволяють ефективно управляти розробкою модульних архітектур за рахунок гнучкості, ітеративності та орієнтації на співпрацю. Вони допомагають швидко реагувати на зміни вимог, забезпечувати високу якість програмного забезпечення та ефективно координувати роботу незалежних команд.

1.3 Специфіка корпоративного простору інжинірингових послуг та вимоги до програмного забезпечення

Корпоративний сектор інжинірингових послуг являє собою складне та динамічне середовище, яке висуває специфічні вимоги до програмного забезпечення. Розробка програмного забезпечення для інжинірингових компаній відрізняється від розробки для інших галузей, і для успішної реалізації проєктів необхідно враховувати ці особливості.

1.3.1 Специфіка інжинірингових послуг:

1. Висока складність проєктів:

- Інжинірингові проєкти часто охоплюють широкий спектр дисциплін, таких як механіка, електротехніка, гідравліка, термодинаміка та інші. Вони можуть включати в себе проєктування складних систем, моделювання фізичних процесів, аналіз великих обсягів даних та управління різноманітними ресурсами. Це вимагає від програмного забезпечення здатності обробляти складні алгоритми, інтегрувати різні типи даних та забезпечувати високу точність розрахунків;

2. Динамічність вимог:

- В інжинірингових проєктах вимоги до програмного забезпечення можуть змінюватися на будь-якому етапі, від початкового проєктування до введення в експлуатацію. Це може бути пов'язано зі змінами в законодавстві,

новими технологічними рішеннями, уточненнями від замовника або непередбаченими обставинами. Програмне забезпечення має бути достатньо гнучким та адаптивним, щоб враховувати ці зміни без значних витрат часу та ресурсів;

3. Важливість точності та надійності:

- В інжинірингових проєктах точність та надійність програмного забезпечення мають критичне значення. Помилки в розрахунках або збої в роботі системи можуть призвести до серйозних наслідків, включаючи фінансові втрати, затримки в реалізації проєктів та навіть загрозу життю та здоров'ю людей. Тому програмне забезпечення має проходити ретельне тестування та відповідати високим стандартам якості;

4. Необхідність інтеграції з іншими системами:

- В інжинірингових компаніях програмне забезпечення рідко використовується ізольовано. Воно має інтегруватися з іншими системами, такими як системи автоматизованого проєктування (CAD), системи управління виробництвом (MES), системи управління проєктами, системи аналізу даних та інші. Це вимагає від програмного забезпечення наявності відкритих інтерфейсів та підтримки стандартів інтеграції;

5. Різноманітність замовників та проєктів:

- Інжинірингові компанії можуть працювати з широким колом замовників, від державних організацій до приватних компаній, і реалізовувати різноманітні проєкти, від проєктування будівель, інфраструктури аеропортів до розробки складних технологічних систем. Це вимагає від програмного забезпечення гнучкості та можливості налаштування під специфічні потреби кожного замовника та проєкту;

1.3.2 Вимоги до програмного забезпечення:

1. Функціональність:

- Програмне забезпечення має забезпечувати повний спектр функцій, необхідних для виконання конкретних інжинірингових завдань. Це може

включати в себе інструменти для проєктування, моделювання, аналізу, оптимізації, візуалізації, управління даними та документацією, співпраці та комунікації;

2. Надійність та безпека:

- Програмне забезпечення має бути надійним в експлуатації, забезпечувати цілісність даних та захист від несанкціонованого доступу. В деяких галузях, таких як авіація та енергетика, вимоги до надійності та безпеки є особливо високими, і програмне забезпечення має відповідати суворим стандартам та нормативним вимогам;

3. Масштабованість:

- Програмне забезпечення має бути здатним обробляти зростаючі обсяги даних та запитів, а також адаптуватися до збільшення кількості користувачів та розширення функціональності. Це особливо важливо для компаній, які планують розвиватися та розширювати свою діяльність;

4. Інтегрованість:

- Програмне забезпечення має мати можливість інтегруватися з іншими системами, що використовуються в компанії, для забезпечення безперебійного обміну даними та автоматизації процесів. Це може включати в себе інтеграцію з CAD/CAM системами, системами управління проєктами, базами даних та хмарними сервісами;

5. Зручність використання:

- Програмне забезпечення має мати інтуїтивно зрозумілий інтерфейс, бути простим у навчанні та використанні. Це допоможе інженерам та іншим співробітникам ефективно виконувати свою роботу та уникнути помилок;

6. Гнучкість та адаптивність:

- Програмне забезпечення має бути гнучким та адаптивним, щоб враховувати зміни вимог, нові технології та різноманітні умови проєктів. Це може бути досягнуто за допомогою модульної архітектури, конфігурованих налаштувань та можливості розширення функціональності;

1.3.3 Вплив специфіки на вибір архітектури:

Специфіка інжинірингових послуг та вимоги до програмного забезпечення безпосередньо впливають на вибір архітектури програмного забезпечення. Модульні архітектури, такі як мікросервіси, дозволяють створювати гнучкі та масштабовані системи, які легко адаптуються до змін вимог. Agile-методології допомагають ефективно управляти розробкою в динамічному середовищі та забезпечувати швидку реакцію на зміни.

Розробка програмного забезпечення для корпоративного простору інжинірингових послуг має свою специфіку та вимагає врахування низки факторів. Модульні архітектури та Agile-методології є ефективним підходом до створення програмного забезпечення, яке відповідає вимогам цього динамічного середовища. Застосування цих підходів дозволяє:

- Підвищити гнучкість та адаптивність програмного забезпечення до змін вимог;
- Спростити розробку та тестування за рахунок розбиття системи на незалежні модулі;
- Забезпечити масштабованість та інтегрованість з іншими системами;
- Підвищити ефективність розробки за рахунок застосування Agile-практик.

РОЗДІЛ 2. ЕФЕКТИВНЕ УПРАВЛІННЯ МОДУЛЬНИМИ АРХІТЕКТУРАМИ В AGILE-СЕРЕДОВИЩІ

У попередньому розділі ми розглянули теоретичні основи модульних архітектур та Agile-методологій, а також специфіку інжинірингових послуг. Тепер перейдемо до практичних аспектів - як ефективно управляти розробкою програмного забезпечення з модульною архітектурою, використовуючи Agile-підходи.

Цей розділ присвячений саме цьому питанню. В ньому ми детально розглянемо ключові аспекти управління модульними архітектурами в Agile-середовищі, зокрема:

1. Планування та організація:

- Як правильно спланувати проєкт з використанням модульної архітектури? Як організувати команди розробників для ефективної роботи над окремими модулями? Як визначити пріоритети та розподілити завдання?

2. Розробка та тестування:

- Які практики та інструменти слід використовувати для розробки та тестування модулів? Як забезпечити якість коду та сумісність модулів між собою? Як організувати процес безперервної інтеграції та доставки (CI/CD)?

3. Впровадження та підтримка:

- Як правильно впроваджувати модульні системи в корпоративне середовище інжинірингової компанії? Як забезпечити їх безперебійну роботу та подальшу підтримку? Як управляти версіями та оновленнями модулів?

4 . Управління залежностями:

- Як ефективно управляти залежностями між модулями? Як запобігти конфліктам та забезпечити цілісність системи при зміні або оновленні модулів?

Цей розділ пропонує практичні рекомендації та інструменти для ефективного управління модульними архітектурами в інжинірингових

компаніях, що дозволить підвищити продуктивність розробки, якість програмного забезпечення та задоволеність замовників.

2.1. Планування та організація розробки модульних систем з використанням Agile-підходів

Ефективне управління розробкою модульних систем в Agile-середовищі вимагає ретельного планування та організації. Ось ключові аспекти, які слід враховувати:

2.1.1 Визначення модулів та їх функціональності:

1. Декомпозиція системи:

- На початку проєкту необхідно ретельно проаналізувати вимоги та розбити систему на незалежні модулі;

2. Функціональність та інтерфейси:

- Для кожного модуля слід чітко визначити його функціональність та інтерфейси взаємодії з іншими модулями;

3. Розмір та гранулярність:

- Важливо знайти баланс між розміром модулів та їх кількістю. Занадто великі модулі втрачають переваги модульності, а занадто малі можуть ускладнити управління залежностями;

2.1.2 Організація команд розробників:

1. Незалежні команди:

- Для кожного модуля або групи пов'язаних модулів слід сформувати незалежну команду розробників (до 10 чоловік у команді);

2. Крос-функціональні команди:

- Команди мають бути крос-функціональними і включати в себе всіх необхідних спеціалістів (розробників, тестувальників, аналітиків тощо);

3. Автономія та відповідальність:

- Команди повинні мати достатню автономію у прийнятті рішень щодо розробки свого модуля, але в той же час нести відповідальність за його якість та інтеграцію з іншими модулями;

2.1.3 Планування ітерацій та релізів:

1. Ітеративний підхід:

- Розробка модулів має здійснюватися в короткі ітерації (спринти) з регулярним випуском робочих версій;

2. Пріоритезація:

- Необхідно визначити пріоритети розробки модулів та їх функціональності на основі бізнес-цінності та залежностей;

3. Синхронізація релізів:

- Важливо спланувати релізи модулів таким чином, щоб забезпечити сумісність та уникнути конфліктів;

2.1.4 Використання Agile-практик:

1. Scrum, Kanban, XP:

- Використання Agile-практик, таких як Scrum, Kanban або XP, допоможе ефективно управляти розробкою модулів;

2. Спільне планування:

- Залучення всіх команд до процесу планування допоможе забезпечити координацію та врахувати залежності між модулями;

3. Демонстрації та ретроспективи:

- Регулярні демонстрації прогресу та ретроспективи дозволять отримувати зворотній зв'язок та вдосконалювати процес розробки.

2.1.5 Інструменти для управління модульними архітектурами:

1. Системи контролю версій:

- Git, SVN тощо для управління кодом модулів та їх версіями;

2. Системи управління залежностями:

- Maven, Gradle, npm тощо для управління залежностями між модулями;

3. Системи безперервної інтеграції/безперервної поставки (CI/CD):

- Jenkins, GitLab CI, CircleCI тощо для автоматизації процесів збірки, тестування та розгортання модулів;

4. Інструменти для співпраці та комунікації:

- Jira, Slack, Microsoft Teams тощо для забезпечення ефективної комунікації між командами;

Планування та організація є критично важливими для успішної розробки модульних систем в Agile-середовищі. Визначення модулів, організація команд, планування ітерацій, використання Agile-практик та інструментів допоможуть ефективно управляти розробкою, забезпечити якість програмного забезпечення та досягти поставлених цілей.

2.2 Розробка, тестування та інтеграція модулів

Управління розробкою, тестуванням та інтеграцією модулів є ключовим аспектом успішного впровадження модульних архітектур. В цьому розділі ми розглянемо найкращі практики та інструменти для забезпечення ефективності цих процесів.

2.2.1 Розробка модулів:

1. Дотримання принципів модульності:

- При розробці модулів важливо дотримуватися принципів інкапсуляції, слабкої зв'язності та високої когезії. Це забезпечить незалежність модулів, спростить їх тестування та підтримку;

2. Використання чітких інтерфейсів:

- Кожен модуль має мати чітко визначений інтерфейс, що описує його функціональність та способи взаємодії з іншими модулями;

3. Врахування залежностей:

- При розробці модулів важливо враховувати залежності між ними та мінімізувати їх кількість;

4. Код-рев'ю:

- Регулярні код-рев'ю допоможуть забезпечити якість коду та дотримання стандартів кодування;

2.2.2 Тестування модулів:

1. Автоматизоване тестування:

- Автоматизоване тестування є необхідним для забезпечення якості модулів та швидкого виявлення помилок;

2. Різні рівні тестування:

- Слід використовувати різні рівні тестування (модульне, інтеграційне, системне) для перевірки функціональності модулів та їх взаємодії;

3. Тестування інтерфейсів:

- Особливу увагу слід приділити тестуванню інтерфейсів між модулями для забезпечення їх сумісності.

4. Навантажувальне тестування:

- Для критично важливих модулів слід проводити навантажувальне тестування для перевірки їх працездатності під високим навантаженням.

2.2.3 Інтеграція модулів:

1. Безперервна інтеграція:

- Часта інтеграція коду модулів в загальний репозиторій допоможе виявляти помилки на ранніх етапах та забезпечувати сумісність;

2. Інтеграційне тестування:

- Після інтеграції модулів слід проводити інтеграційне тестування для перевірки їх спільної роботи;

3. Використання mock-об'єктів:

- При тестуванні модулів можна використовувати mock-об'єкти для імітації роботи інших модулів, від яких вони залежать;

4. Управління версіями:

- Важливо використовувати системи контролю версій для відстеження змін в коді модулів та управління їх версіями.

2.2.4 Інструменти для розробки, тестування та інтеграції:

1. Integrated Development Environment (IDE) з підтримкою модульного тестування: IntelliJ IDEA, Eclipse, Visual Studio тощо;

2. Фреймворки для модульного тестування:

- JUnit, NUnit, pytest тощо;

3. Інструменти для інтеграційного тестування:

- Selenium, Cucumber, SoapUI тощо;

4. Системи безперервної інтеграції/безперервної поставки (CI/CD):

- Jenkins, GitLab CI, CircleCI тощо;

Ефективна розробка, тестування та інтеграція модулів є ключовими факторами успіху при використанні модульних архітектур. Дотримання принципів модульності, автоматизоване тестування, безперервна інтеграція та використання відповідних інструментів допоможуть забезпечити якість програмного забезпечення та прискорити процес розробки.

2.3 Впровадження та підтримка модульних систем в інжинірингових компаніях

Впровадження та підтримка модульних систем в інжинірингових компаніях - це важливий етап, що вимагає ретельного планування та врахування специфіки галузі. Ось ключові аспекти, які слід враховувати:

2.3.1 Планування впровадження:

1. Поетапне впровадження:

- Рекомендується впроваджувати модульну систему поетапно, починаючи з окремих модулів або підсистем. Це дозволить мінімізувати ризики та отримати досвід роботи з новою системою;

2. Визначення пріоритетів:

- Слід визначити пріоритети впровадження модулів на основі їх бізнес-цінності та залежностей;

3. Підготовка інфраструктури:

- Перед впровадженням необхідно переконатися, що інфраструктура компанії готова до роботи з модульною системою. Це може включати в себе налаштування серверів, баз даних, мережі та інших компонентів.

4. Навчання персоналу:

- Співробітники компанії мають бути навчені роботі з новою системою. Це може включати в себе проведення тренінгів, створення інструкцій та документації.

2.3.2 Моніторинг та управління:

1. Моніторинг роботи модулів:

- Після впровадження необхідно забезпечити моніторинг роботи модулів для виявлення помилок та проблем з продуктивністю;

2. Логування та аналіз даних:

- Слід налаштувати систему логування та аналізу даних для отримання інформації про роботу модулів та виявлення потенційних проблем;

3. Управління версіями та оновленнями:

- Важливо стежити за оновленнями модулів та впроваджувати їх вчасно для забезпечення безпеки та актуальності системи;

2.3.3 Підтримка та еволюція:

1. Виправлення помилок:

- Слід забезпечити своєчасне виправлення помилок та усунення проблем, що виникають в процесі експлуатації модульної системи;

2. Технічна підтримка:

- Користувачам системи має бути надана технічна підтримка для вирішення проблем та отримання допомоги в роботі з системою;

3. Еволюція системи:

- Модульна система має бути здатною до еволюції та розвитку для задоволення змінних потреб бізнесу. Це може включати в себе додавання нових модулів, зміну існуючих або інтеграцію з новими системами;

2.3.4 Врахування специфіки інжинірингових компаній:

1. Інтеграція з іншими системами:

- При впровадженні модульної системи важливо враховувати необхідність її інтеграції з іншими системами, що використовуються в інжиніринговій компанії (CAD (Computer - Aided Design, Система автоматизованого проєктування і розрахунку), MES (Manufacturing Execution System, Система управління виробничими процесами), PLM (Product Lifecycle Management, керування життєвим циклом виробу) тощо);

2. Високі вимоги до надійності та безпеки:

- В інжинірингових компаніях часто висуваються високі вимоги до надійності та безпеки програмного забезпечення, оскільки від нього може залежати безпека людей та успіх проєктів;

3. Динамічність вимог:

- Вимоги до програмного забезпечення в інжинірингових проєктах можуть змінюватися протягом всього життєвого циклу проєкту, що вимагає гнучкості та адаптивності модульної системи;

2.3.5 Інструменти для впровадження та підтримки:

1. Системи моніторингу:

- Zabbix, Nagios, Prometheus тощо для моніторингу роботи модулів та інфраструктури;

2. Системи логування:

- ELK stack, Graylog, Splunk тощо для збору та аналізу логів;

3. Системи управління конфігураціями:

- Ansible, Puppet, Chef тощо для автоматизації налаштування та управління інфраструктурою;

4. Системи для співпраці та комунікації: Jira Service Desk, Zendesk, Freshservice тощо для організації технічної підтримки;

Впровадження та підтримка модульних систем в інжинірингових компаніях є складним, але важливим процесом. Ретельне планування, моніторинг, управління версіями, технічна підтримка та врахування специфіки галузі допоможуть забезпечити успішну експлуатацію системи та отримати максимальну віддачу від інвестицій.

2.4 Управління залежностями та забезпечення цілісності

Модульні архітектури, хоч і надають безліч переваг у розробці програмного забезпечення, вносять додатковий рівень складності через наявність залежностей між модулями. Ефективне управління цими залежностями є критично важливим для забезпечення цілісності, стабільності та масштабованості системи. Нехтування цим аспектом може призвести до серйозних проблем, таких як “ефект доміно” при внесенні змін, циклічні залежності, конфлікти версій та зниження загальної надійності системи.

2.4.1 Види залежностей та їх вплив:

Залежності між модулями можуть мати різну природу та проявлятися на різних рівнях. Ось деякі з найпоширеніших видів залежностей:

1. Функціональні залежності:

- Виникають, коли один модуль безпосередньо використовує функціональність, реалізовану в іншому модулі. Наприклад, модуль авторизації може залежати від модуля управління користувачами для перевірки даних аутентифікації. Зміна в модулі управління користувачами (наприклад, формат зберігання паролів) може потребувати відповідних змін в модулі авторизації;

2. Залежності від даних:

- Виникають, коли один модуль очікує отримання даних в певному форматі від іншого модуля або зберігає дані, які використовуються іншим

модулем. Наприклад, модуль аналітики може залежати від даних, зібраних модулем моніторингу. Будь-які зміни в структурі або форматі цих даних можуть порушити роботу модуля аналітики;

3. Залежності від інтерфейсів:

- Виникають, коли модулі взаємодіють між собою через чітко визначені інтерфейси, такі як Application Programming Interface, інтерфейс програмування додатків, програмний інтерфейс програми (API). Зміна інтерфейсу одного модуля може потребувати змін в інших модулях, які його використовують. Важливо забезпечити стабільність інтерфейсів та версійність (API) для мінімізації таких впливів;

4. Залежності від технологій:

- Виникають, коли модулі залежать від конкретних технологій, бібліотек або фреймворків. Наприклад, один модуль може бути написаний на Java, а інший - на Python. Або ж модулі можуть використовувати різні версії однієї і тієї ж бібліотеки, що може призвести до конфліктів;

2.4.2 Проблеми, пов'язані з залежностями та їх наслідки:

Неефективне управління залежностями може призвести до ряду серйозних проблем:

1. “Ефект доміно”:

- Зміна в одному модулі може спричинити ланцюгову реакцію змін в інших модулях, що залежать від нього. Це ускладнює внесення змін, збільшує час та витрати на розробку, а також підвищує ризик виникнення помилок;

2. Циклічні залежності:

- Коли модулі мають циклічні залежності (А залежить від Б, Б залежить від В, а В залежить від А), це створює “замкнуте коло” та ускладнює розуміння системи, її розробку, тестування та внесення змін;

3. Конфлікти версій:

- Використання різних версій одних і тих самих бібліотек або компонентів в різних модулях може призвести до несумісності та помилок в роботі системи;

4. Зниження гнучкості та масштабованості:

- Наявність багатьох залежностей ускладнює модифікацію та розширення системи, а також ускладнює заміну окремих модулів більш ефективними або сучасними рішеннями;

2.4.3 Стратегії та практики ефективного управління залежностями:

Для мінімізації ризиків та проблем, пов'язаних з залежностями, слід застосовувати наступні стратегії та практики:

1. Мінімізація залежностей:

- При проєктуванні системи слід прагнути до мінімізації кількості та складності залежностей між модулями. Це можна досягти шляхом чіткого розмежування функціональності, використання абстракцій та загальних інтерфейсів;

2. Використання чітких інтерфейсів:

- Чітко визначені та стабільні інтерфейси (API) допоможуть зменшити зв'язність між модулями. Це дозволить змінювати внутрішню реалізацію модуля без впливу на інші модулі, які використовують його інтерфейс;

3. Управління версіями:

- Використання систем контролю версій (Git, SVN) та управління залежностями (Maven, Gradle, npm) допоможе відстежувати зміни в коді та залежностях, а також запобігати конфліктам версій;

4. Ізоляція залежностей:

- Використання контейнерів (Docker) або віртуальних середовищ дозволить ізолювати залежності кожного модуля та уникнути конфліктів між ними. Це особливо важливо при використанні різних версій бібліотек або фреймворків в різних модулях;

2.4.4 Забезпечення цілісності системи:

Для забезпечення цілісності системи та її стабільної роботи в умовах модульної архітектури важливо використовувати комплексний підхід, що включає в себе різні практики та інструменти.

1. Автоматизоване тестування:

- Регулярне автоматизоване тестування є фундаментом забезпечення цілісності. Воно дозволяє швидко виявляти помилки та проблеми сумісності на ранніх етапах розробки:

а. Модульне тестування зосереджується на перевірці кожного модуля окремо, ізольовано від інших.

б. Інтеграційне тестування перевіряє взаємодію між модулями та їх сумісність.

в. Системне тестування перевіряє роботу всієї системи в цілому.

Автоматизація цих видів тестування дозволяє швидко отримувати зворотній зв'язок про стан системи та оперативно реагувати на помилки.

2. Контрактне тестування:

- Цей підхід ґрунтується на визначенні чітких “контрактів” між модулями, які описують очікувану поведінку та взаємодію. Контрактне тестування дозволяє перевірити, чи дотримуються модулі цих контрактів, забезпечуючи передбачувану та надійну інтеграцію. Це особливо важливо в мікросервісній архітектурі, де модулі можуть розроблятися різними командами або навіть різними компаніями;

3. Моніторинг та аналіз:

- Постійний моніторинг роботи системи в реальному часі дозволяє оперативно виявляти проблеми та аномалії, які можуть свідчити про порушення цілісності. Аналіз логів, метрик та інших даних моніторингу допоможе визначити причину проблем та вжити необхідних заходів;

4. Версійність та управління конфігураціями:

- Важливо вести облік версій модулів та їх залежностей, а також управляти конфігураціями системи. Це допоможе відстежувати зміни, відкочувати оновлення в разі необхідності та забезпечувати стабільність роботи;

5. Статичний аналіз коду:

- Використання інструментів статичного аналізу коду допоможе виявляти потенційні проблеми з залежностями, такі як циклічні залежності або використання застарілих версій бібліотек, ще на етапі розробки;

2.4.5 Додаткові рекомендації:

1. Документування залежностей:

- Створення чіткої документації, що описує залежності між модулями, допоможе розробникам краще розуміти систему та уникати помилок;

2. Використання єдиного репозиторію для усіх модулів:

- Це спростить управління залежностями та забезпечить узгодженість версій;

3. Обмеження доступу до коду модулів:

- Надання доступу до коду модулів лише уповноваженим розробникам допоможе запобігти несанкціонованим змінам та зберегти цілісність системи;

В цьому розділі ми розглянули різні види залежностей, проблеми, які вони можуть спричинити, та стратегії їх ефективного управління. Слід прагнути до мінімізації кількості та складності залежностей між модулями ще на етапі проєктування системи. Це допоможе зменшити ризик виникнення проблем та спростити розробку та підтримку. Використання чітко визначених та стабільних інтерфейсів є ключовим для зменшення зв'язності між модулями та забезпечення їх незалежної розробки та еволюції. Системи контролю версій та управління залежностями допоможуть відстежувати зміни та запобігати конфліктам версій. Регулярне автоматизоване тестування на всіх рівнях (модульне, інтеграційне, системне) є необхідним для виявлення проблем сумісності та забезпечення цілісності системи. Використання контрактного

тестування допоможе забезпечити правильну взаємодію між модулями та зменшити ризик виникнення проблем при їх інтеграції. Постійний моніторинг роботи системи та аналіз логів дозволять оперативно виявляти та вирішувати проблеми, пов'язані з залежностями.

Дотримання цих рекомендацій допоможе інжиніринговим компаніям ефективно управляти залежностями в модульних системах, забезпечити цілісність та стабільність їх роботи, а також зменшити витрати на розробку та підтримку. Це, в свою чергу, сприятиме підвищенню якості програмного забезпечення, задоволеності замовників та конкурентоспроможності компанії на ринку.

РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ

У попередніх розділах ми розглянули теоретичні основи модульних архітектур та Agile-методологій, а також детально проаналізували ключові аспекти управління розробкою модульних систем в Agile-середовищі. Тепер настав час перевірити ефективність запропонованого підходу на практиці.

Цей розділ присвячений практичному застосуванню та аналізу ефективності розроблених рекомендацій. В ньому ми проведемо наступне:

1. “Case study”:

- Детально проаналізуємо реальний приклад використання модульної архітектури в інжиніринговій компанії на прикладі бізнес-утиліти “WorkHive”. Розглянемо архітектуру системи, процес розробки, виявлені проблеми та досягнуті результати;

2. Моделювання:

- Розробимо модель процесу управління модульною архітектурою в інжиніринговій компанії з урахуванням специфіки галузі та запропонованих рекомендацій. Це дозволить візуалізувати процес та визначити ключові етапи та завдання;

3. Оцінка ефективності:

- Проведемо оцінку ефективності запропонованого підходу до управління модульними архітектурами, використовуючи кількісні та якісні показники. Це допоможе визначити ступінь впливу запропонованих рекомендацій на продуктивність розробки, якість програмного забезпечення та задоволеність замовників;

Результати цього розділу дозволять підтвердити практичну значимість дослідження та ефективність запропонованого підходу до управління модульними архітектурами в інжиніринговых компаніях.

3.1 “Case study”: аналіз використання модульної архітектури на прикладі бізнес-утиліти “WorkHive”

Для практичного застосування та аналізу ефективності запропонованого підходу до управління модульними архітектурами в інжинірингових компаніях проведемо “Case study” на прикладі бізнес-утиліти *“WorkHive”*.

3.1.1 Опис бізнес-утиліти “WorkHive”

“WorkHive” - це модульна бізнес-утиліта, спрямована на ефективне управління проєктами, персоналом та робочим часом в корпоративному середовищі. Вона розробляється з використанням Agile-методологій та призначена для інжинірингових компаній, таких як ТОВ “Прогрестех-Україна”.

3.1.2 Унікальна ціннісна пропозиція (Value proposition):

1. Модульна архітектура:
   - гнучка система, що дозволяє адаптувати *“WorkHive”* під індивідуальні потреби кожного підприємства;
2. Інтуїтивно зрозумілий інтерфейс:
   - легкість використання та швидке освоєння для всіх співробітників;
3. Комплексне відстеженню проєктного часу:
   - точний облік робочого часу та аналіз продуктивності на індивідуальному та командному рівнях;
4. Інструменти мікроменеджменту:
   - можливості для ефективного планування, розподілу завдань та контролю виконання;
5. Інтеграції з іншими системами:
   - сумісність з популярними корпоративними інструментами для більш ефективного управління бізнесом;
6. Аналітика та звітності:

- детальні звіти та аналітика для прийняття обґрунтованих рішень щодо оптимізації робочих процесів;

3.1.3 Tone of Voice

Tone of Voice (ToV) - це важливий аспект, який допоможе сформувати унікальний образ продукту та встановити зв'язок з цільовою аудиторією.

3.1.4 Основні характеристики:

1. Професійний:
   - “WorkHive” - це серйозний інструмент для бізнесу, тому ToV повинен відображати експертність та надійність;
2. Дружній:
   - Водночас, “WorkHive” має бути простим та зручним у використанні, тому ToV повинен бути привітним та доступним;
3. Мотивуючий:
   - “WorkHive” спрямований на підвищення продуктивності та досягнення цілей, тому ToV повинен надихати та заряджати енергією;
4. Позитивний:
   - “WorkHive” допомагає вирішувати проблеми та досягати успіхів, тому ToV повинен бути оптимістичним та підтримуючим.

3.1.5 Формат бізнес-утиліти “WorkHive”

Визначення формату це важлива деталь, яка значно розширить потенційну аудиторію “WorkHive” та зробить її більш доступною для користувачів, тому просто необхідно додати до дорожньої карти проєкту “WorkHive” пункт про кросплатформеність.

3.1.6 Кросплатформеність

“WorkHive” буде доступна на всіх популярних платформах, включаючи:

1. Веб-версія (доступ через браузер з будь-якого корпоративного пристрою (комп'ютер, ноутбук, планшет, смартфон)).
2. Десктопні додатки (окремі додатки для Windows, macOS та Linux для більш зручного та швидкого доступу до функціоналу).
3. Мобільні додатки (додатки для iOS та Android, що дозволять користувачам працювати з “WorkHive” з будь-якого місця та в будь-який час).

3.1.7 Технічна реалізація

Для забезпечення кросплатформеності “WorkHive” ми розглянемо такі варіанти.

Прогресивні веб-додатки (PWA) - це дозволить створити веб-додаток, який поводиться як нативний додаток на різних платформах, забезпечуючи швидкість, офлайн-доступ та інші переваги.

Фреймворки кросплатформеної розробки - використання таких фреймворків, як React Native, Flutter або Xamarin, дозволить створити додатки для різних платформ з єдиної кодової бази, що значно скоротить час та витрати на розробку.

Хмарні технології - розміщення “WorkHive” в хмарі забезпечить доступність з будь-якого пристрою з підключенням до інтернету та спростить оновлення та підтримку продукту.

3.1.8 Переваги:

1. Розширення аудиторії:
   - більше користувачів зможуть отримати доступ до “WorkHive”, незалежно від їхніх корпоративних пристроїв та операційних систем;
2. Зручність використання:
   - користувачі зможуть працювати з “WorkHive” на звичних для них платформах та пристроях;

3. Економія часу та ресурсів:
   - розробка кросплатформеного рішення дозволить скоротити витрати на створення та підтримку окремих додатків для кожної платформи;
4. Швидке оновлення:
   - оновлення продукту будуть доступні для всіх користувачів одночасно, незалежно від їх платформи.

Врахування кросплатформеності на ранніх етапах розробки допоможе нам створити більш універсальний та зручний продукт, який задовольнить потреби широкої аудиторії.

3.1.9 Аналіз архітектури "***WorkHive***"

Утиліта "WorkHive" має модульну архітектуру, що складається з набору незалежних модулів, таких як:

3.1.9.1 Основні функції утиліти:

1. Підвищення продуктивності:

- оптимізація робочих процесів та ефективне використання часу співробітників;

2. Зниження витрат:

- скорочення витрат на оплату праці завдяки точному обліку робочого часу та виявленню неефективних процесів;

3. Покращення прийняття рішень:

- детальна аналітика та звіти допомагають керівникам приймати обґрунтовані рішення;

4. Підвищення залученості співробітників:

- прозорість та контроль над власним часом сприяють підвищенню мотивації та залученості;

3.1.9.2 Основні модулі утиліти:

1. Development Tickets (система відстеження та управління завданнями (тікетами) для розробників).

2. Workdays Calendar (календар робочих днів для планування та відстеження робочого часу співробітників).

3. Approval Report / Unapproved Worktime Report (модуль для створення та затвердження звітів про робочий час, включаючи можливість перегляду непідтверджених звітів).

4. Shift Scheduler (інструмент для планування та управління змінами співробітників).

5. Organization Structure Editor Tool (редактор організаційної структури компанії).

6. Corrections (модуль для внесення виправлень та коригувань у дані системи).

7. Vacation Scheduler (планувальник відпусток для співробітників).

8. Training Portal (портал для навчання та розвитку співробітників).

9. Performance Assessment (інструмент для оцінки ефективності роботи співробітників).

10. Logon Info (інформація про вхід користувача в систему).

11. Active queues (перегляд активних черг завдань).

12. My Workflow (персональний робочий модуль користувача).

13. Employee Project History (історія проєктів, в яких брав участь співробітник).

14. Eager To Help (ця функція для швидкого звернення до служби підтримки або довідкової інформації).

15. FIX IT (ця функція для швидкого звернення до працівників відповідальних за ремонт майна на території офісного простору).

Кожен модуль має чітко визначену функціональність та інтерфейси взаємодії з іншими модулями. Це забезпечує гнучкість та масштабованість

системи, дозволяючи додавати, видаляти або змінювати модулі без впливу на інші частини системи.

3.1.10 Аналіз процесу розробки "WorkHive"

Розробка "WorkHive" здійснюється з використанням Agile-методологій, що дозволяє швидко реагувати на зміни вимог та забезпечувати високу якість програмного забезпечення.

1. Ітеративний підхід:

- Розробка ведеться в короткі ітерації (спринти), кожна з яких завершується випуском робочої версії продукту;

2. Спільна робота:

- Команда розробників тісно співпрацює між собою та із замовником;

3. Безперервна інтеграція та тестування:

- Код регулярно інтегрується та тестується для забезпечення якості та сумісності модулів;

3.1.11 Виявлені проблеми та досягнуті результати

В процесі розробки "WorkHive" були виявлені наступні проблеми:

1. Складність управління залежностями між модулями:

- Для вирішення цієї проблеми було використано чітке визначення інтерфейсів та систем управління залежностями;

2. Необхідність забезпечення сумісності модулів:

- Для цього було впроваджено автоматизоване тестування на різних рівнях (модульне, інтеграційне, системне);

Досягнуті результати:

1. Створено гнучку та масштабовану систему, яка може бути легко адаптована до змінних потреб бізнесу.

2. Забезпечено високу якість програмного забезпечення за рахунок застосування Agile-методологій та автоматизованого тестування.

3. Скорочено час виведення продукту на ринок завдяки ітеративному підходу та безперервній інтеграції.

3.1.12 Висновки "Case study"

Аналіз використання модульної архітектури на прикладі "WorkHive" показав, що запропонований підхід до управління модульними системами в Agile-середовищі є ефективним та дозволяє досягти позитивних результатів.

Далі ми розглянемо моделювання процесу управління модульною архітектурою в інжиніринговій компанії та проведемо оцінку ефективності запропонованого підходу.

3.2 Моделювання процесу управління модульною архітектурою в інжиніринговій компанії

Для глибокого розуміння та ефективного впровадження запропонованого підходу до управління модульними архітектурами в інжинірингових компаніях, важливо не тільки описати його теоретично, але й представити у вигляді наочної моделі. Така модель дозволяє візуалізувати процес, визначити ключові етапи, завдання та їх взаємозв'язок, а також спростити аналіз та оптимізацію процесу.

В цьому розділі ми розробимо деталізовану модель процесу управління модульною архітектурою, адаптовану до специфіки інжинірингових послуг та враховуючи рекомендації, викладені в попередніх розділах.

3.2.1 Етапи процесу та їх детальний опис:

1. Планування:

- Цей етап є фундаментом успішної розробки модульної системи. Він включає в себе наступні кроки:

а. Визначення вимог:

- Детальний аналіз потреб замовника та вимог до системи, включаючи функціональні та нефункціональні вимоги (продуктивність,

надійність, безпека тощо). Важливо врахувати специфіку інжинірингових проєктів, таких як висока складність, динамічність вимог та необхідність інтеграції з іншими системами;

б. Декомпозиція системи:

- Розбиття системи на незалежні модулі з чітким визначенням їх функціональності та відповідальності. При цьому важливо враховувати принципи модульності (інкапсуляція, слабка зв'язність, висока когезія) та знайти оптимальний баланс між розміром модулів та їх кількістю;

в. Визначення залежностей:

- Аналіз та документування залежностей між модулями, включаючи функціональні залежності, залежності від даних та інтерфейсів. Це допоможе уникнути циклічних залежностей та "ефекту доміно" при внесенні змін;

г. Формування команд:

- Створення крос-функціональних команд розробників, відповідальних за розробку окремих модулів або груп пов'язаних модулів. Команди повинні мати достатню автономію та відповідальність за свій модуль;

д. Вибір Agile-методології:

- Вибір найбільш підходящої Agile-методології (Scrum, Kanban, XP тощо) та її адаптація до специфіки проєкту та компанії;

ж. Планування ітерацій та релізів:

- Визначення тривалості ітерацій (спринтів), планування релізів модулів та синхронізація їх між собою для забезпечення сумісності та уникнення конфліктів;

2. Розробка:

- Цей етап включає в себе безпосередню розробку модулів з використанням Agile-практик та інструментів:

а. Розробка модулів:

- Дотримання принципів модульності при розробці коду, використання чітких інтерфейсів для взаємодії між модулями, врахування залежностей та регулярні код-рев'ю для забезпечення якості коду;

б. Автоматизоване тестування:

- Впровадження автоматизованого тестування на різних рівнях (модульне, інтеграційне, системне) для швидкого виявлення помилок та забезпечення якості та сумісності модулів;

в. Безперервна інтеграція та доставка (CI/CD):

- Автоматизація процесів збірки, тестування та розгортання модулів для прискорення розробки та забезпечення швидкого виведення нових версій на ринок;

г. Використання систем контролю версій та управління залежностями:

- Застосування Git, SVN та інших систем для управління кодом та залежностями між модулями, що допоможе відстежувати зміни та запобігати конфліктам версій;

3. Впровадження:

- На цьому етапі модульна система розгортається в продуктивному середовищі інжинірингової компанії:

а. Поетапне впровадження:

- Впровадження модулів поетапно, починаючи з найбільш пріоритетних або найменш залежних, для мінімізації ризиків та отримання досвіду роботи з новою системою;

б. Підготовка інфраструктури:

- Налаштування серверів, баз даних, мережі та інших компонентів інфраструктури для забезпечення стабільної роботи модульної системи;

в. Інтеграція з іншими системами:

- Інтеграція модульної системи з іншими системами, що використовуються в компанії, такими як CAD, MES, PLM тощо;

г. Навчання персоналу:

- Проведення тренінгів та надання документації для навчання співробітників роботі з новою системою;

4. Підтримка:

- Цей етап включає в себе заходи щодо забезпечення безперебійної роботи та подальшого розвитку модульної системи:

а. Моніторинг та аналіз:

- Постійний моніторинг роботи модулів, аналіз логів та метрик для виявлення помилок, проблем з продуктивністю та потенційних ризиків;

б. Виправлення помилок:

- Своєчасне виправлення помилок та усунення проблем, що виникають в процесі експлуатації системи;

в. Управління версіями та оновленнями:

- Впровадження оновлень модулів для забезпечення безпеки, актуальності та розширення функціональності системи;

г. Технічна підтримка:

- Надання технічної підтримки користувачам системи для вирішення проблем та отримання допомоги;

3.2.2 Візуалізація моделі:

Візуалізація моделі процесу управління модульною архітектурою є важливим кроком для її кращого розуміння та аналізу. Графічне представлення процесу дозволяє наочно побачити взаємозв'язок між етапами, завданнями та рішеннями, а також спрощує комунікацію між учасниками проєкту.

Для візуалізації моделі можна використовувати різні діаграми та нотації, кожна з яких має свої переваги та недоліки:

1. BPMN (Business Process Model and Notation):

- BPMN - це стандартний інструмент для моделювання бізнес-процесів;

- Він надає широкий набір елементів для відображення послідовності дій, паралельних процесів, умов, циклів та інших аспектів процесу;

- BPMN-діаграми легко читаються та розуміються завдяки стандартизованим символам та правилам;

- Перевагами є стандартизований інструмент, широкий набір елементів інструментарію, легкість розуміння;

- До недоліків можна віднести те, що даний інструмент може бути занадто детальним для відображення загальної структури процесу;

2. UML (Unified Modeling Language):

- UML - це мова моделювання, що широко використовується в розробці програмного забезпечення;

- Вона надає різні типи діаграм для моделювання різних аспектів системи, включаючи діаграми компонентів, діаграми розгортання, діаграми активності тощо;

- Перевагами є гнучкість та можливість моделювання різних аспектів системи;

- Як недолік, вимагає знання UML-Notations, може бути складною для неспеціалістів;

3. Блок-схеми:

- Блок-схеми - це простий та інтуїтивно зрозумілий спосіб візуалізації процесів.

- Вони використовують блоки різних форм для відображення етапів процесу та стрілки для відображення послідовності дій.

- Перевагами є простота та наочність.

- Недоліками є обмежені можливості для відображення складних аспектів процесу.

Вибір нотації або інструменту залежить від конкретних потреб та переваг. Для візуалізації моделі процесу управління модульною архітектурою в інжиніринговій компанії можна використати BPMN-діаграму для детального відображення процесу або блок-схему для більш загального представлення. UML-діаграми можуть бути використані для моделювання архітектури системи та взаємодії модулів.

Важливо, щоб обрана візуалізація була чіткою, зрозумілою та інформативною для всіх учасників проєкту.

3.2.3 Приклад візуалізації (спрощена блок-схема):

Спрощений приклад візуалізації моделі процесу управління модульною архітектурою у вигляді блок-схеми:

[Початок] --> [Планування]

[Планування] --> [Розробка]

[Розробка] --> [Тестування]

[Тестування] --(ОК)--> [Впровадження]

[Тестування] --(Помилки)--> [Розробка]

[Впровадження] --> [Підтримка]

[Підтримка] --> [Моніторинг]

[Моніторинг] --(Проблеми)--> [Розробка]

[Моніторинг] --(ОК)--> [Закінчення]

Ця блок-схема відображає основні етапи процесу та їх взаємозв'язок.

Розшифровка блоків:

1. Початок:

- Початок процесу розробки;

2. Планування:

- Визначення вимог, декомпозиція на модулі, формування команд тощо;

3. Розробка:

- Створення коду модулів;

4. Тестування:

- Перевірка працездатності та сумісності модулів;

5. Впровадження:

- Розгортання системи в продуктивному середовищі;

6. Підтримка:

- Моніторинг, виправлення помилок, оновлення тощо;

7. Моніторинг:

- Спостереження за роботою системи;

8. Закінчення:

- Завершення процесу розробки (або перехід до нового циклу).

Для більш детальної візуалізації можна використати BPMN-діаграму або іншу нотацію, яка дозволить відобразити більше деталей процесу.

Приклад BPMN-діаграми для детальної візуалізації процесу управління модульною архітектурою в інжиніринговій компанії.

Опис процесу на діаграмі:

1. Процес починається з події “Початок проєкту”.

2. Далі виконується дія “Планування”, яка включає в себе визначення вимог, декомпозицію системи на модулі, формування команд тощо.

3. Після планування розпочинається “Розробка” модулів. Цей процес може бути розбитий на окремі підпроцеси для кожного модуля.

4. Паралельно з розробкою здійснюється “Тестування” модулів.

5. Після успішного тестування модулі “Впроваджуються” в продуктивне середовище.

6. Далі здійснюється “Підтримка” системи, яка включає в себе моніторинг, виправлення помилок та оновлення.

7. Процес завершується подією “Завершення проєкту”.

Розшифровка елементів діаграми:

1. Події:

- Кола відображають події, які ініціюють або завершують процес (початок, закінчення, отримання вимог тощо);

2. Дії:

- Прямокутники відображають дії або завдання, які виконуються в процесі (визначення вимог, розробка модуля, тестування тощо);

3. Шлюзи:

- Ромби відображають точки розгалуження процесу (перевірка умов, паралельне виконання тощо);

4. Потоки:

- Стрілки відображають послідовність виконання дій;

5. Учасники:

- Доріжки відображають учасників процесу (команда розробки, замовник, тестувальники тощо);

3.2.4 Опис ключових елементів моделі:

1. Зворотний зв'язок:

- На кожному етапі процесу важливо отримувати зворотній зв'язок від замовників, користувачів та розробників для вдосконалення процесу та продукту;

2. Agile-принципи:

- Модель базується на Agile-принципах, таких як ітеративність, співпраця, гнучкість та орієнтація на цінність для замовника;

3. Автоматизація:

- Автоматизація процесів тестування, інтеграції та розгортання є важливою складовою ефективного управління модульними архітектурами;

4. Управління ризиками:

- На кожному етапі процесу необхідно виявляти та управляти ризиками, пов'язаними з розробкою та впровадженням модульної системи.

Розроблена модель процесу управління модульною архітектурою в інжинірінговій компанії надає наочне уявлення про ключові етапи та завдання. Вона дозволяє краще зрозуміти процес, визначити потенційні проблеми та розробити стратегії їх вирішення. Ця модель може бути використана як інструмент для планування, організації та контролю розробки модульних систем в інжинірингових компаніях.

3.3 Оцінка ефективності запропонованого підходу

В цьому розділі ми проведемо оцінку ефективності запропонованого підходу до управління модульними архітектурами в інжинірингових компаніях. Для цього ми використаємо як кількісні, так і якісні показники, а

також врахуємо результати “Case study” та моделювання, представлені в попередніх розділах.

3.3.1 Кількісні показники:

1. Час розробки:

- Порівняння часу, витраченого на розробку модульної системи з використанням запропонованого підходу, з часом, який був би витрачений на розробку аналогічної системи з використанням традиційних методів. Очікується, що запропонований підхід дозволить скоротити час розробки за рахунок паралельної розробки модулів, повторного використання коду та автоматизації процесів;

2. Кількість помилок:

- Порівняння кількості помилок, виявлених в модульній системі, розробленій з використанням запропонованого підходу, з кількістю помилок в аналогічних системах, розроблених традиційними методами. Очікується, що запропонований підхід дозволить зменшити кількість помилок за рахунок модульного тестування, безперервної інтеграції та чітких інтерфейсів між модулями;

3. Витрати на розробку:

- Порівняння витрат на розробку модульної системи з використанням запропонованого підходу з витратами на розробку аналогічної системи традиційними методами. Очікується, що запропонований підхід дозволить зменшити витрати за рахунок скорочення часу розробки, зменшення кількості помилок та підвищення ефективності роботи команд;

3.3.2 Якісні показники:

1. Гнучкість та адаптивність:

- Оцінка здатності модульної системи адаптуватися до змін вимог та умов проєкту. Запропонований підхід має забезпечити високу гнучкість за

рахунок можливості легко додавати, видаляти або змінювати модулі без впливу на інші частини системи;

2. Масштабованість:

- Оцінка здатності модульної системи масштабуватися для обробки зростаючих обсягів даних та запитів. Запропонований підхід має забезпечити високу масштабованість за рахунок можливості додавати нові модулі або збільшувати потужність існуючих;

3. Надійність та стабільність:

- Оцінка надійності та стабільності роботи модульної системи. Запропонований підхід має забезпечити високу надійність за рахунок ізоляції модулів, автоматизованого тестування та моніторингу;

4. Задоволеність замовника:

- Оцінка задоволеності замовника функціональністю, якістю та своєчасністю розробки модульної системи. Запропонований підхід має забезпечити високу задоволеність замовника за рахунок залучення його до процесу розробки, швидкої реакції на зміни вимог та високої якості програмного забезпечення.

3.3.3 Методи оцінки:

1. Аналіз даних проєкту:

- Аналіз даних про час розробки, кількість помилок, витрати та інші показники проєкту “WorkHive” (з наданого вами файлу) для оцінки ефективності запропонованого підходу;

2. Опитування експертів:

- Проведення опитування експертів в галузі розробки програмного забезпечення та інжинірингових послуг для отримання їх оцінки ефективності запропонованого підходу;

3. Порівняльний аналіз:

- Порівняння результатів проєкту “WorkHive” з даними про аналогічні проєкти, реалізовані з використанням традиційних методів розробки.

3.3.4 Очікувані результати:

Очікується, що запропонований підхід до управління модульними архітектурами в інжинірингових компаніях дозволить:

- Скоротити час розробки програмного забезпечення;
- Зменшити кількість помилок та підвищити якість програмного забезпечення;
- Зменшити витрати на розробку;
- Підвищити гнучкість, масштабованість та надійність програмного забезпечення;
- Підвищити задоволеність замовників;

Оцінка ефективності є важливим етапом дослідження, який дозволяє підтвердити практичну значимість запропонованого підходу. Використання кількісних та якісних показників, а також різних методів оцінки, допоможе об'єктивно оцінити ефективність запропонованого підходу до управління модульними архітектурами в інжинірингових компаніях.

РОЗДІЛ 4. РОЗРОБКА ЕТАПІВ ПРОЄКТУ (ПРОГРАМИ) ТА ПЕРЕЛІКУ РОБІТ

4.1 Етапи проєкту

Розробка та впровадження “WorkHive” буде здійснюватися за такими послідовними етапами:

Таблиця 4.1 - Етапи та зміст робіт проєкту, що розробляється

Етапи роботи	*Код, назва (зміст робіт)*
1 етап *1.1 Планування та аналіз (2 тижні)*	*1.1.1 Детальне визначення вимог до системи, враховуючи потреби цільової аудиторії та результати аналізу конкурентів.* *1.1.2 Розробка детальної концепції “WorkHive”, включаючи її модульну структуру, функціональність та інтерфейс користувача.* *1.1.3 Створення технічного завдання (ТЗ) з детальним описом вимог до системи та критеріїв приймання.* *1.1.4 Формування команди проєкту та розподіл ролей та відповідальності.* *1.1.5 Складання детального плану проєкту з визначенням термінів, бюджету та ресурсів.* *1.1.6 Аналіз ризиків проєкту.* *1.1.7 Підготовка презентації для стейкхолдерів.* *1.1.8 Презентація для стейкхолдерів.*

Таблиця 4.1 - Етапи та зміст робіт проєкту, що розробляється

Етапи роботи	*Код, назва (зміст робіт)*
	1.1.9 Фіналізація плану проєкту.
2 етап *2.1 Проєктування (4 тижні)*	*2.1.1 Розробка архітектури системи та вибір технологічного стеку.* *2.1.2 Проєктування бази даних та структури даних.* *2.1.3 Розробка прототипів інтерфейсу користувача та їх тестування з потенційними користувачами.* *2.1.4 Створення дизайн-макетів інтерфейсу.*
3 етап *3.1 Розробка (12 тижнів)*	*3.1.1 Розробка серверної частини "WorkHive".* *3.1.2 Розробка клієнтської частини (веб, десктоп, мобільні додатки).* *3.1.3 Інтеграція з іншими системами (за необхідності).* *3.1.4 Розробка ШІ-модуля.* *3.1.5 Регулярне тестування та виправлення помилок.*
4 етап *4.1 Тестування та налагодження (4 тижні)*	*4.1.1 Проведення альфа-тестування із залученням внутрішніх користувачів.* *4.1.2 Проведення бета-тестування із залученням реальних користувачів.* *4.1.3 Виправлення виявлених помилок та недоліків.* *4.1.4 Оптимізація продуктивності та швидкодії системи.*

Таблиця 4.1 - Етапи та зміст робіт проєкту, що розробляється

Етапи роботи	*Код, назва (зміст робіт)*
5 етап *5.1 Впровадження (2 тижні)*	*5.1.1 Розгортання "WorkHive" на серверах компанії або в хмарі.* *5.1.2 Навчання співробітників роботі з системою.* *5.1.3 Запуск системи в експлуатацію.*
6 етап *6.1 Підтримка та розвиток (постійно)*	*6.1.1 Моніторинг роботи системи, виявлення та виправлення помилок.* *6.1.2 Виправлення помилок та вразливостей.* *6.1.3 Збір та аналіз зворотного зв'язку від користувачів.* *6.1.4 Розробка та впровадження нових функцій та покращень.* *6.1.5 Оновлення документації та навчальних матеріалів.* *6.1.6 Технічна підтримка.*

4.2 Перелік робіт

Детальний перелік робіт для кожного етапу проєкту буде розроблено на основі технічного завдання та плану проєкту. Він включатиме конкретні завдання, терміни виконання, відповідальних осіб та необхідні ресурси.

Важливо усвідомлювати, що зазначені терміни є орієнтовними та можуть бути скориговані в процесі реалізації проєкту.

4.3 Команда проєкту

Таблиця 4.2 - Організація виконання проєктних робіт

Код роботи	*Перелік робіт*	*Організаційна одиниця (виконавець)*
1.1.1	*Детальне визначення вимог до системи, враховуючи потреби цільової аудиторії та результати аналізу конкурентів.*	*Керівник проєкту (Project Manager), Бізнес аналітик,*
1.1.2	*Розробка детальної концепції "WorkHive", включаючи її модульну структуру, функціональність та інтерфейс користувача.*	*Керівник проєкту (Project Manager), Бізнес аналітик, UI/UX дизайнер*
1.1.3	*Створення технічного завдання (ТЗ) з детальним описом вимог до системи та критеріїв приймання.*	*Бізнес аналітик, Керівник проєкту (Project Manager), Технічний спеціаліст*
1.1.4	*Формування команди проєкту та розподіл ролей та відповідальності.*	*Керівник проєкту (Project Manager), HR-менеджер*
1.1.5	*Складання детального плану проєкту з визначенням термінів, бюджету та ресурсів.*	*Керівник проєкту (Project Manager), Бізнес аналітик, Фінансовий менеджер*
1.1.6	*Аналіз ризиків проєкту.*	*Керівник проєкту (Project Manager), Бізнес аналітик*
1.1.7	*Підготовка презентації для стейкхолдерів.*	*Керівник проєкту (Project Manager),*

Таблиця 4.2 - Організація виконання проєктних робіт

Код роботи	*Перелік робіт*	*Організаційна одиниця (виконавець)*
		Бізнес аналітик, UI/UX дизайнер
1.1.8	*Презентація для стейкхолдерів.*	*Керівник проєкту (Project Manager), Бізнес аналітик*
1.1.9	*Фіналізація плану проєкту*	*Керівник проєкту (Project Manager)*
2.1.1	*Розробка архітектури системи та вибір технологічного стеку.*	*Технічний директор, Архітектор програмного забезпечення, Команда розробників*
2.1.2	*Проєктування бази даних та структури даних.*	*Архітектор баз даних, Розробник баз даних*
2.1.3	*Розробка прототипів інтерфейсу користувача та їх тестування з потенційними користувачами.*	*UI/UX дизайнер, Бізнес аналітик , Користувачі*
2.1.4	*Створення дизайн-макетів інтерфейсу.*	*UI/UX дизайнер, Графічний дизайнер*
3.1.1	*Розробка серверної частини "WorkHive".*	*Backend розробники, Архітектор програмного забезпечення*
3.1.2	*Розробка клієнтської частини (веб, десктоп, мобільні додатки).*	*Frontend розробники, UI/UX дизайнер*

Таблиця 4.2 - Організація виконання проєктних робіт

Код роботи	*Перелік робіт*	*Організаційна одиниця (виконавець)*
3.1.3	*Інтеграція з іншими системами (за необхідності).*	*Команда розробників, Системний адміністратор*
3.1.4	*Розробка ШІ-модуля.*	*ШІ-розробник*
3.1.5	*Регулярне тестування та виправлення помилок.*	*QA інженери*
4.1.1	*Проведення альфа-тестування із залученням внутрішніх користувачів.*	*QA інженери, Команда розробників*
4.1.2	*Проведення бета-тестування із залученням реальних користувачів.*	*QA інженери, Команда розробників, Користувачі*
4.1.3	*Виправлення виявлених помилок та недоліків.*	*Команда розробників*
4.1.4	*Оптимізація продуктивності та швидкодії системи.*	*Команда розробників, DevOps інженер*
5.1.1	*Розгортання "WorkHive" на серверах компанії або в хмарі.*	*DevOps інженер, Системний адміністратор*
5.1.2	*Навчання співробітників роботі з системою.*	*HR-менеджер, Керівник проєкту (Project Manager), Бізнес аналітик*
5.1.3	*Запуск системи в експлуатацію.*	*Керівник проєкту (Project Manager),*

Таблиця 4.2 - Організація виконання проєктних робіт

Код роботи	*Перелік робіт*	*Організаційна одиниця (виконавець)*
		Команда розробників, QA інженери
6.1.1	*Моніторинг роботи системи, виявлення та виправлення помилок.*	*DevOps інженер, Служба технічної підтримки*
6.1.2	*Виправлення помилок та вразливостей.*	*Керівник проєкту (Project Manager)*
6.1.3	*Збір та аналіз зворотного зв'язку від користувачів.*	*Бізнес аналітик, Служба технічної підтримки, Менеджер продукту*
6.1.4	*Розробка та впровадження нових функцій та покращень.*	*Команда розробників, Менеджер продукту*
6.1.5	*Оновлення документації та навчальних матеріалів.*	*Технічний письменник, Бізнес аналітик*
6.1.6	*Технічна підтримка.*	*Служба технічної підтримки*

4.3.1 Розподіл ролей та відповідальності

Кожен член команди має чітко визначені ролі та відповідальність, що забезпечує ефективну співпрацю та досягнення цілей проєкту. Керівник проєкту відповідає за координацію роботи команди та комунікацію зі стейкхолдерами. Бізнес-аналітик відповідає за визначення вимог та розробку концепції. Дизайнер відповідає за створення зручного та привабливого інтерфейсу. Розробники відповідають за реалізацію функціоналу.

Тестувальники забезпечують якість продукту. DevOps інженер відповідає за інфраструктуру та розгортання.

Склад команди може варіюватися залежно від масштабу та складності проєкту. Наприклад, для невеликих проєктів деякі ролі можуть бути об'єднані, тоді як для великих проєктів може знадобитися залучення додаткових фахівців (наприклад, маркетологів, технічних письменників тощо).

4.3.2 Висновки щодо формування команди проєкту "WorkHive":

1. Міждисциплінарний підхід:
    - для успішної реалізації проєкту "WorkHive" необхідна команда, що складається з фахівців різних галузей (бізнес-аналітиків, дизайнерів, розробників, тестувальників та DevOps інженерів). Такий підхід забезпечить комплексний підхід до розробки продукту та врахування всіх аспектів його функціональності, зручності використання та технічної реалізації;
2. Agile-методологія:
    - команда повинна бути готова працювати за Agile-методологією, яка передбачає гнучке планування, ітеративну розробку та постійний зворотний зв'язок. Це дозволить швидко реагувати на зміни вимог та забезпечити максимальну цінність для користувачів;
3. Досвід та експертиза:
    - члени команди повинні мати досвід у своїх галузях та бути знайомими з сучасними технологіями та інструментами розробки програмного забезпечення. Це забезпечить високу якість продукту та ефективність роботи команди;
4. Командний дух та комунікація:

    - важливо створити в команді атмосферу взаємоповаги, довіри та відкритості. Ефективна комунікація між членами команди є ключовим фактором успіху проєкту;

4.3.3 Канали пошуку фахівців:

1. Внутрішні ресурси (ТОВ “Прогрестех-Україна” має власний IT-відділ, тому можна розглянути можливість залучення внутрішніх фахівців до проєкту. Це може бути більш економічно вигідним варіантом, а також сприятиме розвитку компетенцій співробітників).
2. Рекрутингові агентства (звернення до спеціалізованих рекрутингових агентств може допомогти знайти кваліфікованих фахівців з необхідним досвідом та навичками).
3. Онлайн-платформи для пошуку роботи (такі платформи, як LinkedIn, Indeed, DOU, Work.ua, дозволяють розміщувати вакансії та шукати резюме фахівців).
4. Соціальні мережі та професійні спільноти (активне використання соціальних мереж (Facebook, Twitter) та професійних спільнот (наприклад, спільноти розробників на GitHub) може допомогти знайти талановитих фахівців, які зацікавлені в участі в інноваційних проєктах).
5. Рекомендації (звернення до колег, партнерів та інших контактів може допомогти знайти фахівців за рекомендаціями).

4.3.4 Умови залучення фахівців:

1. Конкурентна заробітна плата (пропонування конкурентної заробітної плати та бонусів може залучити талановитих фахівців).
2. Цікавий проєкт (робота над інноваційним проєктом, таким як “WorkHive”, може бути додатковою мотивацією для фахівців).
3. Можливості для професійного розвитку (надання можливостей для навчання, підвищення кваліфікації та кар'єрного зростання може зробити проєкт більш привабливим для потенційних кандидатів).
4. Гнучкий графік роботи та віддалена робота (можливість працювати віддалено або за гнучким графіком може бути важливим фактором для деяких фахівців).

5. Корпоративна культура (позитивна та підтримуюча атмосфера в команді, а також цінності компанії, можуть бути додатковим стимулом для приєднання до проєкту).

Враховуючи ці фактори та використовуючи різні канали пошуку, ТОВ “Прогрестех-Україна” зможе сформувати сильну та мотивовану команду, яка успішно реалізує проєкт “WorkHive”.

РОЗДІЛ 5. КАЛЕНДАРНЕ ПЛАНУВАННЯ ПРОЄКТУ

5.1 Оцінка строків та ресурсів проєкту

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти “WorkHive”	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
1			*Етап*	*14 днів* *1.06.24-14.06.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ*
	1.1		*Планування та аналіз*		
		1.1.1	*Детальне визначення вимог до системи*	*3 дні* *1.06.24-3.06.24*	*К, БА*
		1.1.2	*Розробка детальної концепції “WorkHive”*	*3 дні* *4.06.24-6.06.24*	*К, БА, UI/UX*

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
		1.1.3	*Створення технічного завдання (ТЗ)*	*2 дні 7.06.24-8.06.24*	*БА, К, ТС*
		1.1.4	*Формування команди проєкту*	*1 день 9.06.24*	*К, HR*
		1.1.5	*Складання детального плану проєкту*	*1 день 10.06.24*	*К, БА, ФМ*
		1.1.6	*Аналіз ризиків проєкту*	*1 день 11.06.24*	*К, БА*
		1.1.7	*Підготовка презентації для стейкхолдерів*	*1 день 12.06.24*	*К, БА, UI/UX*
		1.1.8	*Презентація для стейкхолдерів*	*1 день 13.06.24*	*К, БА*
		1.1.9	*Фіналізація плану проєкту*	*1 день 14.06.24*	*К*
2			*Етап*	*28 днів*	

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
	2.1		*Проєктування*	*15.06.24-12.07.24*	*К, БА, ТД, АР, АБД, РБД, UI/UX, Кор., ГД*
		2.1.1	*Розробка архітектури системи та вибір технологічного стеку*	*7 днів 15.06.24-21.06.24*	*ТД, АР, КР*
		2.1.2	*Проєктування бази даних та структури даних*	*7 днів 22.06.24-28.06.24*	*АБД, РБД*
		2.1.3	*Розробка прототипів інтерфейсу користувача та їх тестування з потенційними користувачами*	*7 днів 29.06.24-5.07.24*	*UI/UX, БА, Кор.*
		2.1.4	*Створення дизайн-макетів інтерфейсу*	*7 днів*	*UI/UX, ГД*

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
				6.07.24-12.07.24	
3			*Етап*	*84 днів* *13.07.24-4.10.24*	*B-end, АР, F-end, UI/UX, КР, СА, ШІр, QA*
	3.1		*Розробка*		
		3.1.1	*Розробка серверної частини "WorkHive"*	*28 днів* *13.07.24-9.08.24*	*B-end, АР*
		3.1.2	*Розробка клієнтської частини (веб, десктоп, мобільні додатки)*	*28 днів* *10.08.24-6.09.24*	*F-end, UI/UX*
		3.1.3	*Інтеграція з іншими системами*	*14 днів* *7.09.24-20.09.24*	*КР, СА*
		3.1.4	*Розробка ШІ-модуля*	*14 днів* *21.09.24-4.10.24*	*ШІр*

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
		3.1.5	*Регулярне тестування та виправлення помилок*	*84 днів 13.07.24-4.10.24*	*QA*
4			*Етап*	*28 днів 5.10.24-2.11.24*	*QA, КР, Кор., DevOps*
	4.1		*Тестування та налагодження*		
		4.1.1	*Проведення альфа-тестування із залученням внутрішніх користувачів*	*10 днів 5.10.24-14.10.24*	*QA, КР*
		4.1.2	*Проведення бета-тестування із залученням зовнішніх користувачів*	*7 днів 15.10.24-21.10.24*	*QA, КР, Кор.*
		4.1.3	*Виправлення виявлених помилок та недоліків*	*7 днів 22.10.24-28.10.24*	*КР*

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
		4.1.4	*Оптимізація продуктивності та швидкодії системи*	*4 дні 29.10.24-2.11.24*	*КР, DevOps*
5			*Етап*	*14 днів 3.11.24-16.11.24*	*DevOps, СА, HR, К, БА, QA*
	5.1		*Впровадження*		
		5.1.1	*Розгортання "WorkHive" на серверах компанії або в хмарі*	*5 днів 3.11.24-7.11.24*	*DevOps, СА*
		5.1.2	*Навчання співробітників роботі з системою*	*4 дні 8.11.24-11.11.24*	*HR, К, БА*
		5.1.3	*Запуск системи в експлуатацію*	*5 днів 12.11.24-16.11.24*	*К, КР, QA*
6			*Етап*	*Постійно*	

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
	6.1		*Підтримка та розвиток*		*DevOps, СТ, КР, БА, МП, ТП*
		6.1.1	*Моніторинг роботи системи та виявлення помилок*	*Постійно*	*DevOps, СТ*
		6.1.2	*Виправлення помилок та вразливостей*	*Постійно*	*КР*
		6.1.3	*Збір та аналіз зворотного зв'язку від користувачів*	*Постійно*	*БА, СТ, МП*
		6.1.4	*Розробка та впровадження нових функцій та покращень*	*Постійно*	*КР, МП*
		6.1.5	*Оновлення документації та навчальних матеріалів*	*Постійно*	*ТП, БА*

Таблиця 5.1 - Оцінка строків та ресурсів проєкту

№ 1-го рівня	*№ 2-го рівня*	*№ 3-го рівня*	*Назва задачі*	*Оцінка тривалості*	*Ресурси**
			Розробка бізнес-утиліти "WorkHive"	*168 дні* *1.06.24-16.11.24*	*К, БА, Кор., UI/UX, ТС, HR, ФМ, ТД, АР, АБД, РБД, ГД, B-end, F-end, СА, ШІр, QA, DevOps, СТ, МП, ТП*
		6.1.6	*Технічна підтримка користувачів*	*Постійно*	*СТ*

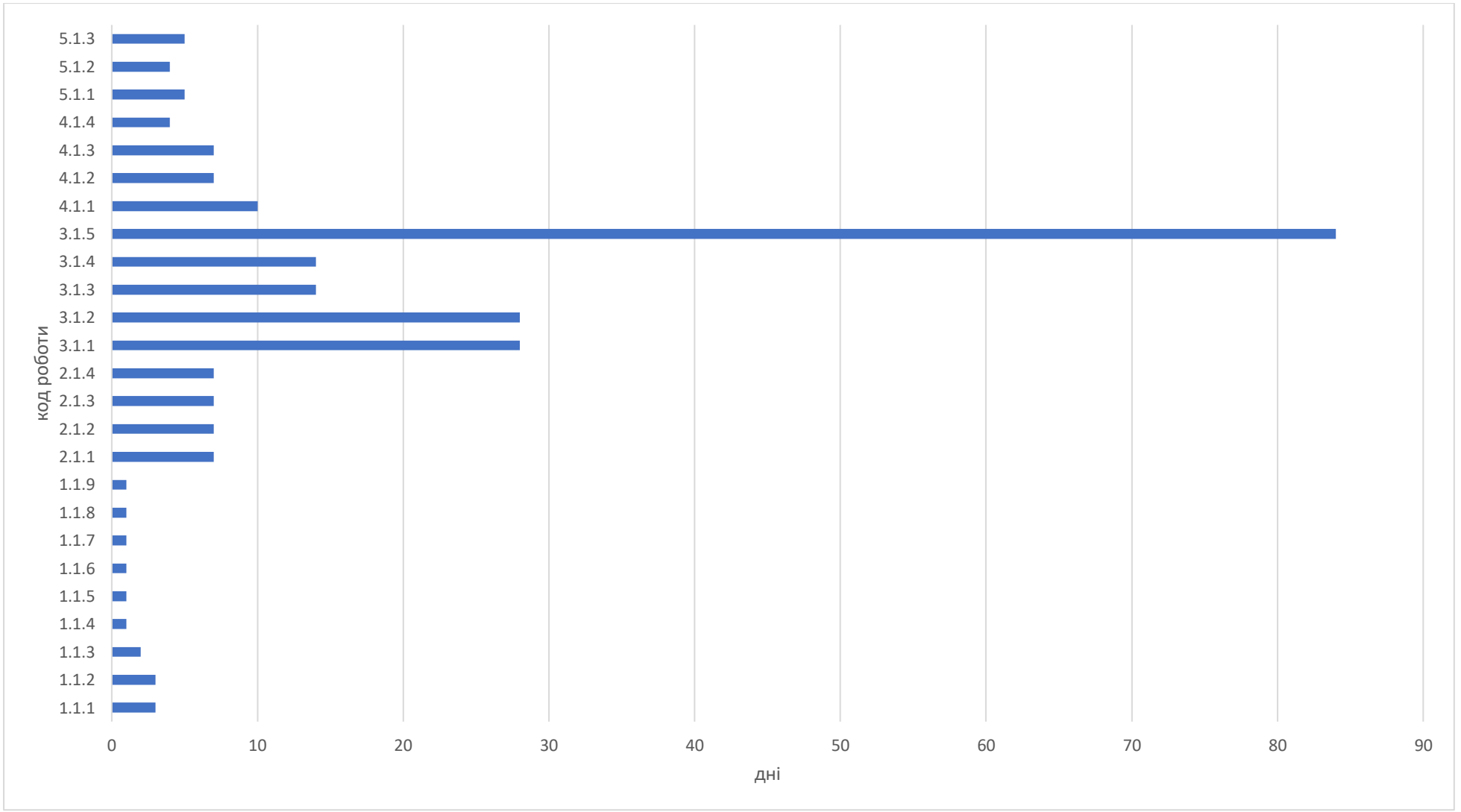
Легенда:

1. К - Керівник проєкту;
2. БА - Бізнес аналітик;
3. Кор - Користувачі;
4. UI/UX - UI/UX дизайнер;
5. ТС - Технічний спеціаліст;
6. HR - HR-менеджер;
7. ФМ - Фінансовий менеджер;
8. ТД - Технічний директор;
9. АР - Архітектор програмного забезпечення;
10. КР - Команда розробників;
11. АБД - Архітектор бази даних;
12. РБД - Розробники баз даних;
13. ГД - Графічний дизайнер;
14. B-end - Backend розробники;
15. F-end - Frontend розробники;
16. СА - Системний адміністратор;
17. ШІр - ШІ-розробники;
18. QA - QA інженери;
19. DevOps - DevOps інженер;
20. СТ - Служба технічної підтримки;
21. МП - Менеджер продукту;
22. ТП - Технічний письменник;

Для візуалізації поточного плану та графіку робіт над проєктом використаємо Діаграму Ганта та балочну діаграму. Як бачимо на малюнках 5.1 і

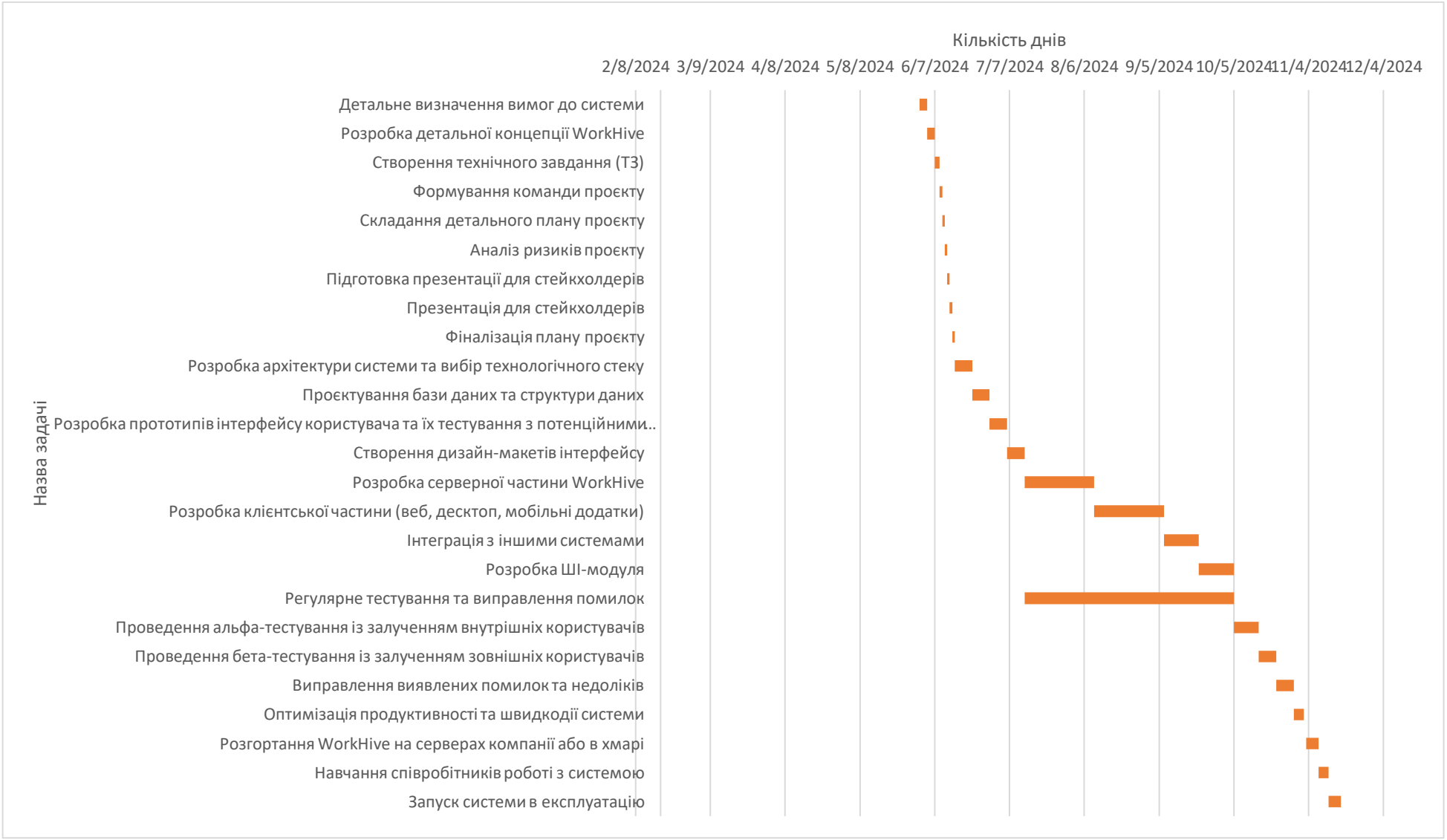
5.2, ці інструменти візуалізації є наочним джерелом такої проєктної інформації:

1. Які роботи є критичними, а які - некритичними.
2. Який запас часу мають некритичні роботи.
3. Коли мають розпочинатися і завершуватися за планом проєктні роботи.
4. Якими є логічні зв’язки між роботами.
5. Яким є фактичне виконання робіт на певну дату.



Малюнок 5.1 - Оцінка тривалості задач

Джерело: розроблено автором



Малюнок 5.2 - Діаграма Ганта

Джерело: розроблено автором

РОЗДІЛ 6. РЕСУРСНЕ ЗАБЕЗПЕЧЕННЯ ПРОЄКТУ

6.1 Загальна потреба в ресурсах

Ми обчислимо загальну людино-днів для кожного завдання, помноживши необхідну кількість людей на день на тривалість завдання. Потім ми підсумуємо ці значення для всіх завдань, щоб отримати загальну трудомісткість (Т).

Трудомісткість роботи (Т) для проєкту “WorkHive” становить 685 людино-днів.

Корисний фонд часу одного працівника ($Ф_{кор}$) - це кількість людино-годин, які працівник може ефективно відпрацювати за певний період. Для розрахунку $Ф_{кор}$ необхідно врахувати наступні фактори:

1. Тривалість робочого дня (зазвичай це 8 годин, але може варіюватися в залежності від компанії та законодавства).
2. Кількість робочих днів на тиждень (зазвичай це 5 днів, але може бути і 6 днів).
3. Кількість робочих тижнів на місяць (це 4 тижні).
4. Відпустки та святкові дні (необхідно відняти дні відпусток та святкових днів, які працівник не працює).
5. Непередбачувані відсутності (необхідно врахувати можливі лікарняні, відгули тощо).

Розрахунок $Ф_{кор}$ для ТОВ “Прогрестех-Україна”:

Припустимо, що:

1. Тривалість робочого дня (8 годин).
2. Кількість робочих днів на тиждень (5 днів).
3. Кількість робочих тижнів на місяць (4 тижні).
4. Відпустки та святкові дні (20 днів на рік (в середньому 1,67 дня на місяць)).
5. Непередбачувані відсутності (2 дні на місяць).

Тоді корисний фонд часу одного працівника за місяць становитиме:

$$\Phi_{\text{кор}} = \left(8\frac{\text{годин}}{\text{день}} * 5\frac{\text{днів}}{\text{тиждень}} * 4\frac{\text{тижні}}{\text{місяці}}\right) - 1.67\frac{\text{дня}}{\text{місяць}} - 2\frac{\text{дні}}{\text{місяць}}$$
$$= 156.33\frac{\text{людино} - \text{годин}}{\text{місяць}}$$

Оскільки проєкт триває 168 днів (приблизно 5.6 місяців), то загальний корисний фонд часу одного працівника за весь проєкт становитиме:

$$\Phi_{\text{кор}} = 156.33\frac{\text{людино} - \text{годин}}{\text{місяць}} * 5.6 \text{ місяців} = 876.45 \text{ людино} - \text{годин}$$

Фактичний корисний фонд часу може відрізнятися залежно від конкретних умов роботи та індивідуальних особливостей працівників. Для більш точного розрахунку необхідно враховувати дані про фактичну відпрацьовану кількість годин та відсутності співробітників.

$$К_р = \frac{\Phi_{\text{кор}}}{Т} = \frac{685}{156.33} = 5$$

Отже нам потрібно 5 людей для виконання роботи за місяць.

Враховуючи обмеження часу, бюджету та використання ресурсів ТОВ “Прогрестех-Україна”, нам необхідно ретельно проаналізувати та оптимізувати використання ресурсів на проєкті “WorkHive”.

Обмеженість простору впливає на максимальну кількість співробітників, які можуть одночасно працювати над проєктом в офісі компанії. Рішенням є оптимізація робочого простору для забезпечення комфортної роботи максимальної кількості співробітників, наприклад, робота в дві зміни. З 7:00 до 15:00 (І зміна) і зміна з графіком, 15:00 до 23:00. Таке планування графіку роботи спрямоване, на те щоб уникнути переповнення офісу. Додатково такий графік вирішує питання з обмеженням кількісті комп'ютерів яке могло б обмежити кількість співробітників, які можуть одночасно працювати над проєктом, особливо якщо потрібне спеціалізоване програмне забезпечення або обладнання.

Також мажемо розглянути можливості віддаленої роботи для деяких членів команди, особливо тих, чия присутність в офісі не є обов'язковою.

Обмежений доступ до об'єкта може ускладнити доступ до необхідних матеріалів та обладнання, що може призвести до затримок у роботі. Рішенням цього має бути ретельне планування та координація з відповідними відділами ТОВ “Прогрестех-Україна” для забезпечення своєчасного доступу до необхідних ресурсів. Необхідно заздалегідь узгоджувати графіки роботи та потреб у матеріалах та обладнанні.

Обмежена кількість комп'ютерів може обмежити кількість співробітників, які можуть одночасно працювати над проєктом, особливо якщо потрібне спеціалізоване програмне забезпечення або обладнання.

Планування графіку роботи таким чином, щоб забезпечити доступ до необхідних ресурсів для всіх членів команди.

Вимоги безпеки праці обмежують можливість роботи у позаурочний час, що може вплинути на терміни виконання проєкту. Ретельне планування робочого часу та завдань, допоможе уникнути необхідності позаурочної роботи.

Продуктивність залежить від ефективності керівництва. Низька ефективність керівництва може призвести до демотивації команди та зниження продуктивності. Рішенням цього є призначення досвідченого та компетентного керівника проєкту, який зможе ефективно керувати командою та мотивувати її до досягнення цілей.

Забезпечення регулярного зворотного зв'язку та підтримки з боку керівництва, створить позитивну та продуктивну атмосферу в команді.

6.1.1 Людські ресурси

Таблиця 6.1 - Визначення потреби у людських ресурсах

Код роботи	*Вид ресурсу*	*Необхідна кількість у день, чол.*	*Тривалість використання ресурсу*	*Термін початку використання ресурсу*
1.1.1	*Керівник проєкту*	*1*	*3 дні*	*1.06.24*
	Бізнес аналітик	*1*	*3 дні*	*1.06.24*

Таблиця 6.1 - Визначення потреби у людських ресурсах

Код роботи	*Вид ресурсу*	*Необхідна кількість у день, чол.*	*Тривалість використання ресурсу*	*Термін початку використання ресурсу*
1.1.2	*Керівник проєкту*	*1*	*3 дні*	*4.06.24*
	Бізнес аналітик	*1*	*3 дні*	*4.06.24*
	UI/UX дизайнер	*3*	*3 дні*	*4.06.24*
1.1.3	*Бізнес аналітик*	*1*	*2 дні*	*7.06.24*
	Керівник проєкту	*1*	*2 дні*	*7.06.24*
	Технічний спеціаліст	*1*	*2 дні*	*7.06.24*
1.1.4	*Керівник проєкту*	*1*	*1 день*	*9.06.24*
	HR-менеджер	*1*	*1 день*	*9.06.24*
1.1.5	*Керівник проєкту*	*1*	*1 день*	*10.06.24*
	Бізнес аналітик	*1*	*1 день*	*10.06.24*
	Фінансовий менеджер	*1*	*1 день*	*10.06.24*
1.1.6	*Керівник проєкту*	*1*	*1 день*	*11.06.24*
	Бізнес аналітик	*1*	*1 день*	*11.06.24*
1.1.7	*Керівник проєкту*	*1*	*1 день*	*12.06.24*
	Бізнес аналітик	*1*	*1 день*	*12.06.24*
	UI/UX дизайнер	*3*	*1 день*	*12.06.24*
1.1.8	*Керівник проєкту*	*1*	*1 день*	*13.06.24*
	Бізнес аналітик	*1*	*1 день*	*13.06.24*
1.1.9	*Керівник проєкту*	*1*	*1 день*	*14.06.24*
2.1.1	*Технічний директор*	*1*	*7 днів*	*15.06.24*
	Архітектор програмного забезпечення	*1*	*7 днів*	*15.06.24*
	Команда розробників	*5*	*7 днів*	*15.06.24*
2.1.2	*Архітектор баз даних*	*1*	*7 днів*	*22.06.24*

Таблиця 6.1 - Визначення потреби у людських ресурсах

Код роботи	*Вид ресурсу*	*Необхідна кількість у день, чол.*	*Тривалість використання ресурсу*	*Термін початку використання ресурсу*
	Розробник баз даних	*1*	*7 днів*	*22.06.24*
2.1.3	*UI/UX дизайнер*	*3*	*7 днів*	*29.06.24*
	Бізнес аналітик	*1*	*7 днів*	*29.06.24*
2.1.4	*UI/UX дизайнер*	*3*	*7 днів*	*6.07.24*
	Графічний дизайнер	*1*	*7 днів*	*6.07.24*
3.1.1	*Backend розробник*	*1*	*28 днів*	*13.07.24*
	Архітектор програмного забезпечення	*1*	*28 днів*	*13.07.24*
3.1.2	*Frontend розробник*	*1*	*28 днів*	*10.08.24*
	UI/UX дизайнер	*1*	*28 днів*	*10.08.24*
3.1.3	*Керівник проєкту*	*1*	*14 днів*	*7.09.24*
	Системний адміністратор	*1*	*14 днів*	*7.09.24*
3.1.4	*ШІ-розробник*	*1*	*14 днів*	*21.09.24*
3.1.5	*QA інженер*	*1*	*84 дні*	*13.07.24*
4.1.1	*QA інженер*	*1*	*10 днів*	*5.10.24*
	Команда розробників	*1*	*10 днів*	*5.10.24*
4.1.2	*QA інженери*	*1*	*7 днів*	*15.10.24*
	Команда розробників	*1*	*7 днів*	*15.10.24*
	Користувачі	*20*	*7 днів*	*15.10.24*
4.1.3	*Команда розробників*	*5*	*7 днів*	*22.10.24*
4.1.4	*Команда розробників*	*5*	*4 дні*	*29.10.24*
	DevOps інженер	*1*	*4 дні*	*29.10.24*
5.1.1	*DevOps інженер*	*1*	*5 днів*	*3.11.24*

Таблиця 6.1 - Визначення потреби у людських ресурсах

Код роботи	*Вид ресурсу*	*Необхідна кількість у день, чол.*	*Тривалість використання ресурсу*	*Термін початку використання ресурсу*
	Системний адміністратор	*1*	*5 днів*	*3.11.24*
5.1.2	*HR-менеджер*	*1*	*4 дні*	*8.11.24*
	Керівник проєкту	*1*	*4 дні*	*8.11.24*
5.1.3	*Керівник проєкту*	*1*	*5 днів*	*12.11.24*
	Команда розробників	*5*	*5 днів*	*12.11.24*
	QA інженер	*1*	*5 днів*	*12.11.24*
6.1.1	*DevOps інженер*	*1*	*Постійно*	*12.11.24*
	Служба технічної підтримки	*2*	*Постійно*	*12.11.24*
6.1.2	*Команда розробників*	*2*	*Постійно*	*12.11.24*
6.1.3	*Бізнес аналітик*	*1*	*Постійно*	*12.11.24*
	Служба технічної підтримки	*2*	*Постійно*	*12.11.24*
	Менеджер продукту	*1*	*Постійно*	*12.11.24*
6.1.4	*Команда розробників*	*2*	*Постійно*	*12.11.24*
	Менеджер продукту	*1*	*Постійно*	*12.11.24*
6.1.5	*Технічний письменник*	*1*	*Постійно*	*12.11.24*
	Бізнес аналітик	*1*	*Постійно*	*12.11.24*
6.1.6	*Служба технічної підтримки*	*2*	*Постійно*	*12.11.24*

Таблиця 6.2 - Календар та обсяги наявних ресурсів

Вид ресурсу	*Наявна кількість у день, чол.*	*Дата початку*	*Дата кінця*
Керівник проєкту	*1*	*3.06.24*	*20.06.24*
Бізнес аналітик	*1*	*3.06.24*	*18.06.24*
UI/UX дизайнер	*1*	*4.06.24*	*12.06.24*
Технічний спеціаліст	*1*	*7.06.24*	*10.06.24*
HR-менеджер	*1*	*10.06.24*	*10.06.24*
Фінансовий менеджер	*1*	*10.06.24*	*10.06.24*
Технічний директор	*1*	*21.06.24*	*2.07.24*
Архітектор програмного забезпечення	*1*	*21.06.24*	*2.07.24*
Команда розробників	*5*	*21.06.24*	*2.07.24*
Архітектор баз даних	*1*	*1.07.24*	*10.07.24*
Розробник баз даних	*1*	*1.07.24*	*10.07.24*
UI/UX дизайнер	*1*	*11.07.24*	*30.07.24*
Бізнес аналітик	*1*	*11.07.24*	*22.07.24*
Графічний дизайнер	*1*	*23.07.24*	*31.07.24*
Backend розробник	*1*	*1.08.24*	*6.09.24*
Архітектор програмного забезпечення	*1*	*1.08.24*	*6.09.24*
Frontend розробник	*1*	*9.08.24*	*17.10.24*
UI/UX дизайнер	*1*	*9.08.24*	*17.10.24*
Керівник проєкту	*1*	*18.10.24*	*14.11.24*
Системний адміністратор	*1*	*18.10.24*	*14.11.24*
ШІ-розробник	*1*	*15.11.24*	*5.12.24*
QA інженер	*1*	*1.08.24*	*2.01.25*
Команда розробників	*5*	*9.12.24*	*3.01.25*

Таблиця 6.2 - Календар та обсяги наявних ресурсів

Вид ресурсу	*Наявна кількість у день, чол.*	*Дата початку*	*Дата кінця*
DevOps інженер	*1*	*30.12.24*	*13.01.25*
Системний адміністратор	*1*	*7.01.25*	*13.01.25*
HR-менеджер	*1*	*21.01.25*	*23.01.25*
Керівник проєкту	*1*	*24.01.25*	*5.02.25*
Команда розробників	*5*	*30.01.25*	*5.02.25*
QA інженер	*1*	*30.01.25*	*5.02.25*

Для визначення, координації та зіставляння видів людського ресурсу із наявними ресурсами організації згідно з календарним планом було взято до уваги:

1. Нормальну продуктивність праці (з урахуванням рівня підготовки і кваліфікації).
2. Існуючі зобов’язання (стосовно інших проєктів, так-як більшість спеціалістів беруться з вже наявного персоналу ТОВ “Прогрестех-Україна”).
3. Очікуваний рівень невиходів (через хвороби та інші причини, припустим, що це буде становити близько 25 %).

Ці всі фактори дали змогу виявити, що попередня дата запуску готового продукту переноситься на 36 робочих днів вперед, відповідно новою датою фіналізації проєкту буде 5.02.25.

Таблиця 6.3 - Потреба у фахівцях

Код Роботи	*Назва задачі*	*Дата початку*	*Дата кінця*	*Необхідний ресурсний день, чол.*
1.1.1	*Детальне визначення вимог до системи*	*3.06.24*	*5.06.24*	*2*
1.1.2	*Розробка детальної концепції "WorkHive"*	*6.06.24*	*10.06.24*	*3*
1.1.3	*Створення технічного завдання (ТЗ)*	*11.06.24*	*12.06.24*	*3*
1.1.4	*Формування команди проєкту*	*13.06.24*	*13.06.24*	*2*
1.1.5	*Складання детального плану проєкту*	*14.06.24*	*14.06.24*	*3*
1.1.6	*Аналіз ризиків проєкту*	*17.06.24*	*17.06.24*	*2*
1.1.7	*Підготовка презентації для стейкхолдерів*	*18.06.24*	*18.06.24*	*3*
1.1.8	*Презентація для стейкхолдерів*	*19.06.24*	*19.06.24*	*2*
1.1.9	*Фіналізація плану проєкту*	*20.06.24*	*20.06.24*	*1*
2.1.1	*Розробка архітектури системи та вибір технологічного стеку*	*21.06.24*	*2.07.24*	*7*
2.1.2	*Проєктування бази даних та структури даних*	*3.07.24*	*11.07.24*	*2*
2.1.3	*Розробка прототипів інтерфейсу користувача та їх тестування з потенційними користувачами*	*12.07.24*	*23.07.24*	*2*
2.1.4	*Створення дизайн-макетів інтерфейсу*	*24.07.24*	*1.08.24*	*2*
3.1.1	*Розробка серверної частини "WorkHive"*	*2.08.24*	*10.09.24*	*2*

Таблиця 6.3 - Потреба у фахівцях

Код Роботи	*Назва задачі*	*Дата початку*	*Дата кінця*	*Необхідний ресурсний день, чол.*
3.1.2	*Розробка клієнтської частини (веб, десктоп, мобільні додатки)*	*11.09.24*	*21.10.24*	*2*
3.1.3	*Інтеграція з іншими системами*	*22.10.24*	*11.11.24*	*2*
3.1.4	*Розробка ШІ-модуля*	*12.11.24*	*2.12.24*	*1*
3.1.5	*Регулярне тестування та виправлення помилок*	*2.08.24*	*2.12.24*	*1*
4.1.1	*Проведення альфа-тестування із залученням внутрішніх користувачів*	*3.12.24*	*17.12.24*	*2*
4.1.2	*Проведення бета-тестування із залученням зовнішніх користувачів*	*18.12.24*	*27.12.24*	*2*
4.1.3	*Виправлення виявлених помилок та недоліків*	*30.12.24*	*9.01.25*	*5*
4.1.4	*Оптимізація продуктивності та швидкодії системи*	*10.01.25*	*16.01.25*	*6*
5.1.1	*Розгортання "WorkHive" на серверах компанії або в хмарі*	*17.01.25*	*23.01.25*	*2*
5.1.2	*Навчання співробітників роботі з системою*	*24.01.25*	*29.01.25*	*2*
5.1.3	*Запуск системи в експлуатацію*	*30.01.25*	*5.02.25*	*7*

Згідно з приведеною Таблицею 6.3 «Потреба у фахівцях» ми можемо бачити чітко розпланований графік виконання задач по створенню проєкта з чітко встановленою кількістю необхідних фахівців.

6.1.2 Технічні ресурси

1. Сервери та хмарна інфраструктура для розробки, тестування та розгортання “WorkHive”.
2. Комп'ютери.
3. Програмне забезпечення для розробників та тестувальників.
4. Ліцензії на програмне забезпечення та інструменти розробки.

6.1.3 Матеріальні ресурси

1. Офісне приміщення та обладнання на базі ТОВ “Прогрестех-Україна”.

6.1.4 Фінансові ресурси

1. Оплата праці співробітників.
2. Оплата послуг підрядників.
3. Витрати на маркетинг та просування продукту.
4. Витрати на ліцензії та програмне забезпечення.

РОЗДІЛ 7. КОШТОРИС ПРОЄКТУ ТА ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ

7.1 Розрахунок кошторису проєкту “WorkHive”

Складання кошторису проєкту дозволить підрахувати та співставити доходи та витрати по проєкту та оцінки його економічної ефективності.

Додаток А Таблиця 7.1 відображає наступні пункти:

1. Витрати на ліцензування, хмарні технології та програмне забезпечення.
2. Витрати на персонал (сюди входять заробітна плата, податки та збори, пов’язані з роботою персоналу, залученого до проєкту).
3. Витрати на обладнання (сюди входять витрати на придбання обладнання та інструментів, необхідних для реалізації проєкту).
4. Інші витрати (сюди входять витрати на послуги сторонніх організацій, які залучаються до реалізації проєкту, наприклад, послуги аутсорс консультантів з бухгалтерії, юристів і маркетологів).

7.2 Визначення чистої теперішньої вартості

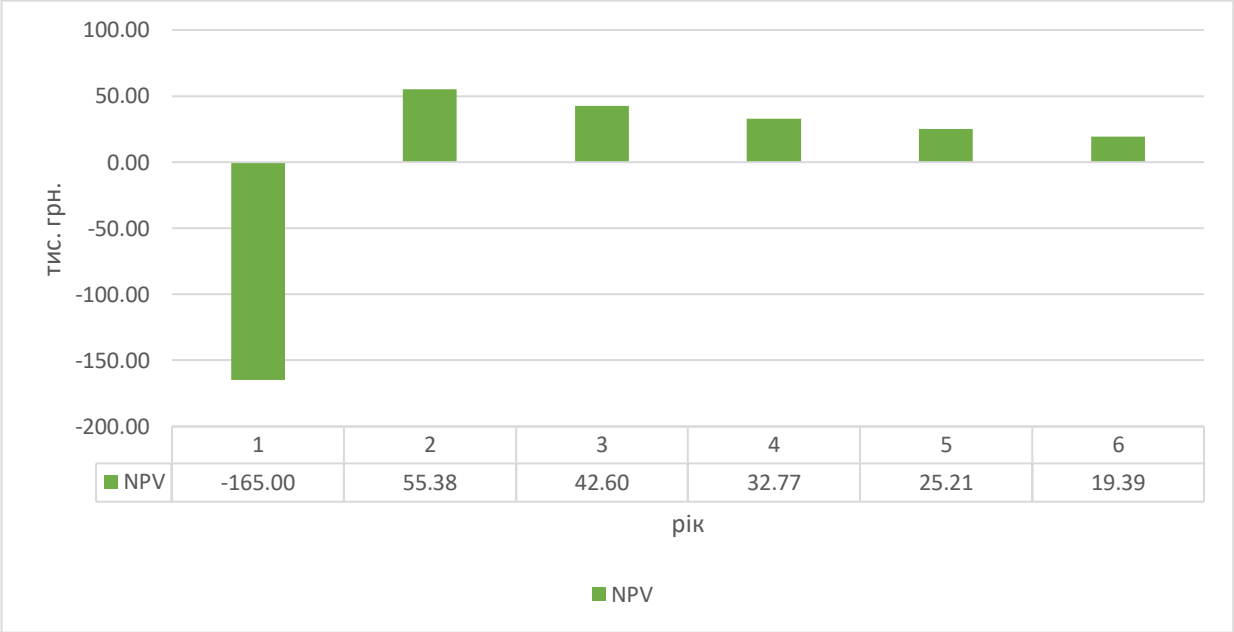
Таблиця 7.2 - Визначення чистої теперішньої вартості

Рік	*К диск.*	*Доходи, тис. грн.*		*Витрати тис. грн.*		*NPV*
		фактичні	*приведені*	*фактичні*	*приведені*	
1	*1*	*85*	*85,00*	*250*	*250,00*	*-165,00*
2	*0,77*	*85*	*65,38*	*13*	*10*	*55,38*
3	*0,59*	*85*	*50,30*	*13*	*7,69*	*42,60*
4	*0,46*	*85*	*38,69*	*13*	*5,92*	*32,77*
5	*0,35*	*85*	*29,76*	*13*	*4,55*	*25,21*
6	*0,27*	*85*	*22,89*	*13*	*3,50*	*19,39*
Всього	*X*	*520*	*292,02*	*315*	*281,66*	*10,36*

BCR: 1.04

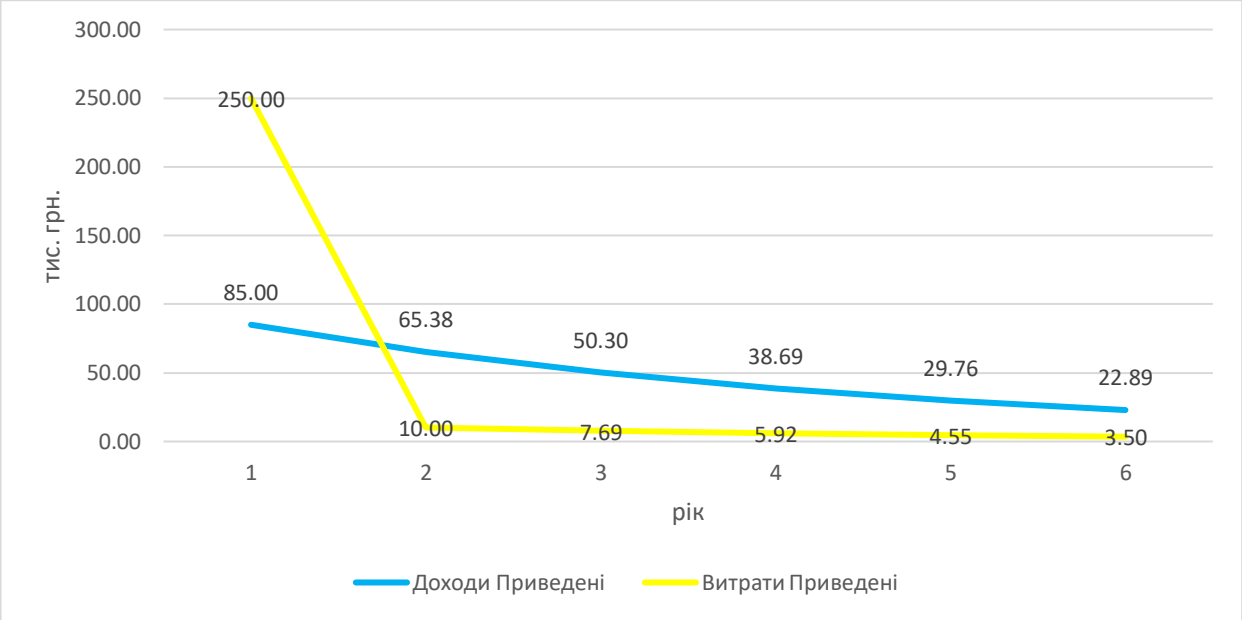
Доходи: 195 тис. грн.

Прибуток від реалізації даного управлінського рішення складає 195 тисяч гривень, тобто доходи перевищують витрати на 195 тис. грн. Разом з цим чиста теперішня вартість даного управлінського рішення складає 10.36 тис. грн. Іншими словами підприємству можна приймати це управлінське рішення по придбанню додаткової одиниці техніки, оскільки приведені доходи (майбутні доходи в теперішніх цінах) вищі за приведені витрати на 10.36 тис. грн. Це також підтверджує й коефіцієнт співвідношення доходів та витрат, який склав 1.04, тобто рівняється трохи вище одиниці, отже і прийняття цього управлінського рішення (запровадження проєкту) є економічно доцільне. Розрахунки відображені в формі діаграм на Малюнки 8.1 і 8.2, які дають можливість візуалізувати показники NVP за вказаний період часу та співставити приведені доходи до витрат.



Малюнок 7.1 - Показник NVP за 6 років

Джерело: розроблено автором



Малюнок 7.2 - Співставлення приведених доходів до витрат

Джерело: розроблено автором

7.3 Визначення внутрішньої норми прибутку

Таблиця 7.3 - Вихідні дані

Роки проєкту	*Вигоди проєкту тис. грн.*	*Витрати проєкту тис. грн.*
1	*105*	*350*
2	*115*	*30*
3	*125*	*25*
4	*145*	*20*
5	*170*	*15*
6	*185*	*11*
Всього	*845*	*451*

Таблиця 7.4 - Розрахунок NPV при ставці дисконту 20%

Рік	*К диск.*	*Доходи*		*Витрати*		*NPV*
		фактичні	*приведені*	*фактичні*	*приведені*	
1	*1*	*105*	*85,00*	*350*	*350,00*	*-265,00*
2	*0,83*	*115*	*95,83*	*30*	*25,00*	*70,83*
3	*0,69*	*125*	*86,81*	*25*	*17,36*	*69,44*
4	*0,58*	*145*	*83,91*	*20*	*11,57*	*72,34*
5	*0,48*	*170*	*81,98*	*15*	*7,23*	*74,75*
6	*0,40*	*185*	*74,35*	*11*	*4,42*	*69,93*
Всього	*X*	*845*	*507,88*	*451*	*415,59*	*92,29*

Таблиця 7.5 - Розрахунок NPV при ставці дисконту 30%

Рік	*К диск.*	*Доходи*		*Витрати*		*NPV*
		фактичні	*приведені*	*фактичні*	*приведені*	
1	*1*	*105*	*85,00*	*350*	*350,00*	*-265,00*
2	*0,77*	*115*	*88,46*	*30*	*23,08*	*65,38*
3	*0,59*	*125*	*73,96*	*25*	*14,79*	*59,17*
4	*0,46*	*145*	*66,00*	*20*	*9,10*	*56,90*
5	*0,35*	*170*	*59,52*	*15*	*5,25*	*54,27*
6	*0,27*	*185*	*49,83*	*11*	*2,96*	*46,86*
Всього	*X*	*845*	*422,77*	*451*	*405,19*	*17,59*

IRR=20+92,29*(30-20) / (92,29-17,59))= 32,26%

Отже ми отримали значення IRR яке дорівнює 32,26%, тобто це та гранична ставка, яку ми можемо заплатити за позичені кошти. Це значення є рівнем беззбитковості проєкту.

ВИСНОВКИ

У кваліфікаційній роботі було проведено комплексне дослідження ефективного управління модульними архітектурами в рамках Agile-методологій, орієнтоване на специфіку інжинірингових компаній. Використання модульної архітектури дозволяє створювати гнучкі програмні рішення, що відповідають високим вимогам корпоративного простору, де необхідна адаптивність до змін і швидке реагування на нові бізнес-запити.

Проведений аналіз переваг та недоліків модульних архітектур підтвердив доцільність їх використання для створення надійних, масштабованих і легко підтримуваних систем. Agile-методології, такі як Scrum і Kanban, сприяють ефективному управлінню розробкою таких архітектур, забезпечуючи ітеративність процесу і можливість адаптації на кожному етапі. Розроблені рекомендації з планування, тестування та впровадження модулів сприяють підвищенню якості кінцевого продукту та зниженню витрат на підтримку.

На основі аналізу кейсу «WorkHive» було розроблено та оцінено модель управління модульною архітектурою, яка дозволяє забезпечити структуровану та керовану інтеграцію модулів у корпоративне середовище. Модель охоплює всі етапи розробки, від планування до підтримки та оновлення системи, що дозволяє зменшити ризики та підвищити якість продукту. Оцінка ефективності цього підходу вказує на позитивний вплив на продуктивність, зниження витрат на підтримку та підвищення задоволеності клієнтів.

Результати дослідження підтверджують, що впровадження модульної архітектури в корпоративному просторі інжинірингових послуг за допомогою Agile-методологій є ефективним підходом, що забезпечує адаптивність, ефективність процесів розробки та підтримки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сайт компанії ТОВ “Прогрестех-Україна” URL: https://progresstech.ua/ (дата звернення: 4.10.2024.).
2. Сайт компанії LLC “Boeing” URL: https://www.boeing.com/ (дата звернення: 4.10.2024.).
3. Сайт Української асоціації IT-професіоналів: https://itukraine.org.ua/
4. Стан українського IT-ринку 2023 URL: https://dou.ua/lenta/articles/it-export-3-quarter-2023/
5. IT-індустрія в Україні 2023, AIN.UA, URL: https://ain.ua/special/book-of-ukrainian-tech-2022-h1/
6. Технології в Україні: Огляд, Deloitte, URL: https://www2.deloitte.com/ua/uk/footerlinks/newsroom/deloitte-research.html
7. Магічний квадрант для гнучких інструментів планування, Gartner, URL: https://www.gartner.com/en/research/methodologies/magic-quadrants-research
8. The Forrester Wave™: Enterprise Agile Planning Tools, Q2 2023, Forrester, URL: https://www.smartsheet.com/platform
9. Гнучке управління проєктами: вичерпний посібник, Atlassian, URL: https://www.atlassian.com/agile
10. Як використовувати ШІ в управлінні проєктами, URL: https://flexi-project.com/uk/%D1%8F%D0%BA-%D0%B2%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%BE%D0%B2%D1%83%D0%B2%D0%B0%D1%82%D0%B8-%D0%B0%D1%96-%D0%B2-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%96-%D0%BF%D1%80/
11. Sutherland, J. (2014). Scrum: The Art of Doing Twice the Work in Half the Time. New York: Crown Business.
12. Ries, E. (2011). *The Lean Startup.* New York: Crown Business.
13. Patton, J. (2014). *User Story Mapping.* Sebastopol, CA: O'Reilly Media.

14. Алькема В. Г., Кириченко О. С. (2023). Менеджмент організацій: навчальний посібник. Кн.1. Київ: Університет «КРОК». 276 с. Електронний ресурс КРОК. URL: https://library.krok.edu.ua/ua/kategoriji/navchalni-posibniki/1440-menedzhment-orhanizatsii
15. Сайт “COLOBRIDGE Blog” URL: https://blog.colobridge.net/uk/2024/01/what-is-microservices-architecture-ua/
16. Сайт Scrum.org URL: https://www.scrum.org/resources/what-scrum-module
17. Сайт Wikipedia.org URL: https://en.wikipedia.org/wiki/Kanban_board

ДОДАТОК А

Розглянемо кошторис проєкту, що дозволить підрахувати та співставити доходи та витрати по проєкту та оцінки його економічної ефективності, Таблиця 7.1.

Таблиця 7.1 - Кошторис проєкту "WorkHive"

№	*Найменування витрат*	*Дедлайн*	*Заплановані витрати*		
			Вартість за одиницю (UAH)	*Загальна вартість (UAH)*	*Загальна вартість (USD)*
1	*Розробка програмного середовища*			*1533120*	*38328*
1.1	*Ліцензії на програмне забезпечення*	*Місяці*	*259*	*93120*	*2328*
1.1.1	*IDE (інтегровані середовища розробки IntelliJ IDEA Ultimate)*	*12*	*167*	*60000*	*1500*
1.1.2	*Фреймворки та бібліотеки*	*12*	*0*	*0*	*0*
1.1.3	*Інструменти для тестування та налагодження (JetBrains dotCover, dotTrace, dotMemory)*	*12*	*67*	*24000*	*600*

Таблиця 7.1 - Кошторис проєкту "WorkHive"

№	Найменування витрат	Дедлайн	Заплановані витрати		
			Вартість за одиницю (UAH)	Загальна вартість (UAH)	Загальна вартість (USD)
1.1.4	Інструменти для управління версіями коду - GitHub;	12	5	1920	48
1.1.5	Інструменти для безперервної інтеграції та доставки (CI/CD) – CircleCI;	12	20	7200	180
1.2	Хмарні сервіси AWS (Amazon Web Services)	36	1333	1440000	36000
2	Заробітна плата команди розробників та підтримки			6898886	172472
2.1	Команда розробників	Дні	52051	1138886	28472
2.1.1	Керівник проєкту (Project Manager)	37	3809	140 952	3524
2.1.2	Бізнес-аналітик	15	2857	42857	1071
2.1.3	UI/UX дизайнер	46	2381	109526	2738
2.1.4	Технічний спеціаліст	2	3571	7142	179
2.1.5	HR-менеджер	5	1905	9525	238
2.1.6	Фінансовий менеджер	1	1429	1429	36
2.1.7	Технічний директор	7	3429	24003	600

Таблиця 7.1 - Кошторис проєкту "WorkHive"

№	Найменування витрат	Дедлайн	Заплановані витрати		
			Вартість за одиницю (UAH)	Загальна вартість (UAH)	Загальна вартість (USD)
2.1.8	Архітектор програмного забезпечення	35	3333	116655	2916
2.1.9	Команда розробників (5 чоловік)	33	4405	145365	3634
2.1.10	Архітектор баз даних	7	3095	21665	542
2.1.11	Розробник баз даних	7	4933	34531	863
2.1.12	Графічний дизайнер	7	1905	13335	333
2.1.13	Backend розробник	28	2857	79996	2000
2.1.14	Frontend розробник	28	2619	73332	1833
2.1.15	Системний адміністратор	14	1904	26656	666
2.1.16	ШІ-розробник	14	3095	43330	1083
2.1.17	QA інженер	106	2143	227158	5679
2.1.18	DevOps інженер	9	2381	21429	536
2.2	Команда підтримки	Місяці	7620	5760000	144000
2.2.1	DevOps інженер	36	2381	1800000	45000
2.2.2	Служба технічної підтримки (2 людини)	36	1429	1080000	27000
2.2.3	Менеджер продукту	36	2143	1620000	40500
2.2.4	Технічний письменник	36	1667	1260000	31500

Таблиця 7.1 - Кошторис проєкту "WorkHive"

№	Найменування витрат	Дедлайн	Заплановані витрати		
			Вартість за одиницю (UAH)	Загальна вартість (UAH)	Загальна вартість (USD)
3.	Витрати на апаратне забезпечення			2074000	51850
3.1	Сервери та сховища даних	Кількість		260000	6500
3.1.1	Сервер високого рівня	1	100000	100000	2500
3.1.2	Сховище даних (NAS)	4	40000	160000	4000
3.2	Робочі станції	Кількість		1050000	26250
3.2.1	Ноутбуки	20	35000	750000	18750
3.2.2	Комп'ютери	10	30000	300000	7500
3.3	Периферійні пристрої	Кількість		700000	17500
3.3.1	Монітори	30	20000	600000	15000
3.3.2	Клавіатури	10	5000	50000	1250
3.3.3	Миші	30	5000	50000	1250
3.4	Мережеве обладнання	Кількість		64000	1600
3.4.1	Комутатори	2	16000	32000	800
3.4.2	Маршрутизатори	4	8000	32000	800
4.	Інші витрати			1219200	88800
4.1	Юридичні та бухгалтерські послуги	Місяців		2592000	64800

Таблиця 7.1 - Кошторис проєкту "WorkHive"

№	Найменування витрат	Дедлайн	Заплановані витрати		
			Вартість за одиницю (UAH)	Загальна вартість (UAH)	Загальна вартість (USD)
4.1.1	Юридичні Консультації	36	1333	1440000	36000
4.1.2	Ведення бухгалтерії	36	1067	1152000	28800
4.2	Маркетинг	Місяці		960000	24000
4.2.1	Просування бізнес утиліти серед корпоративному середовищі	12	2667	960000	24000
Загальні витрати				**11725206**	**351450**
10% непередбачуваних витрат				**1172521**	**35145**
Бюджет проєкту				**18000000**	**450000**