ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Автоматизація документування ІТ-проєктів»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Іванько Артем Олександрович
Керівник: Флегантов Леонід Олексійович
Рецензент: Ковальчук Станіслав Богданович

Полтава – 2024 року

ВСТУП

Актуальність дослідження. Документування є невід'ємною складовою успішного управління ІТ-проєктами. Воно забезпечує збереження знань, фіксацію процесів і рішень, сприяє передачі інформації між командами та запобігає втратам важливих даних. Однак, у сучасних реаліях, коли проєкти в ІТ-індустрії швидко розвиваються, ручне документування стає неефективним через велику кількість даних і обмежений час. Це створює проблєму несвоєчасного або неповного документування, що може призвести до помилок у проєктах, втрати важливих даних або невідповідності очікувань замовників.

Автоматизація документування є ключовим фактором у підвищенні ефективності та якості проєктної документації. Вона дозволяє значно знизити навантаження на працівників, забезпечити актуальність даних та їх синхронізацію з реальними процесами в проєкті. За допомогою автоматизованих систем можна створювати, оновлювати та зберігати документацію без зайвих ручних операцій, що сприяє зменшенню людських помилок і економії часу.

Автоматизація документування ІТ-проєктів є важливою через постійне зростання складності ІТ-проєктів і необхідність підтримки високих стандартів управління документацією. Це питання має значну вагу для ІТ-індустрії, оскільки його вирішення дозволяє компаніям підвищити продуктивність, зменшити витрати на управління документами та забезпечити відповідність вимогам клієнтів. Автоматизація процесів документування допомагає не лише зберігати інформацію, але й покращує загальну координацію та контроль за проєктом.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0123U105060.

Метою роботи побудова ефективної моделі автоматизації ІТ-проєктів.

Завдання роботи:

1. Розглянути основні підходи до автоматизації процесів документування в IT-проєктах;

2. Проаналізувати сучасні інструменти та технології, що забезпечують ефективну автоматизацію документування;

3. Розробити практичні рекомендації щодо впровадження автоматизації документування для підвищення точності та актуальності документації в IT-проєктах.

Об'єкт дослідження: процеси документування в IT-проєктах.

Предмет дослідження: методи та технології автоматизації документування в IT-проєктах.

Методи дослідження: аналіз науково-технічної літератури та інформаційних джерел з автоматизації документування, емпіричне дослідження процесів документування в IT-проєктах, моделювання автоматизованих процесів створення, оновлення та підтримки документації, порівняльний аналіз інструментів для автоматизації та оцінка їх ефективності.

Інформаційна база дослідження: науково-технічна література; наукові статті та публікації з тематики автоматизації документування, доступні у базах даних, таких як Google Scholar, IEEE Xplore, SpringerLink, ScienceDirect, Scopus, ResearchGate; вітчизняні та зарубіжні стандарти документування й управління проєктами; офіційна документація та ресурси GitLab CI, Jenkins, Sphinx, Swagger, MkDocs; аналітичні матеріали експертів у галузі автоматизації документування та управління IT-проєктами.

Елементи наукової новизни даної роботи полягають у розробці комплексної моделі автоматизації процесу документування IT-проєктів, яка включає інтеграцію сучасних інструментів для CI/CD, генерації та публікації документації. Також новизна полягає в застосуванні науково обґрунтованого підходу до впровадження автоматизації з акцентом на зниження кількості помилок, підвищення швидкості оновлення документації та зменшення витрат на її підтримку.

Практична значущість: результати роботи можуть бути використані для розробки та впровадження автоматизованих систем документування в реальних ІТ-проєктах, що дозволить суттєво підвищити ефективність управління проєктами. Запропонована модель автоматизації може бути адаптована до різних середовищ розробки, що забезпечить актуальність і точність документації на всіх етапах життєвого циклу проєкту.

Апробація результатів дослідження. За результатами проведеного дослідження опубліковано тези доповідей: «Сучасні інструменти автоматизованого документування ІТ-проєктів», Матер. XXI щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІ ЕУП та ІТ ПДАУ», 20 листопада 2024 року, м. Полтава; «Безкоштовні рішення для автоматизованого документування ІТ-проєктів». XLIX International scientific and practical conference «New Areas of Scientific Research: Exploring New Frontiers» (November 27-29, 2024) Naples, Italy. International Scientific Unity, 2024.

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 73 сторінках, містить 18 рисунків і 24 таблиці. Список використаних джерел налічує 68 найменувань.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ТА МЕТОДОЛОГІЧНІ АСПЕКТИ АВТОМАТИЗАЦІЇ ДОКУМЕНТУВАННЯ ІТ-ПРОЄКТІВ

1.1 Документування в ІТ-проєктах

1.1.1 Роль і значення документування в управлінні ІТ-проєктами

Документування в ІТ-проєктах відіграє важливу роль для координації, контролю та забезпечення прозорості проєктних процесів. Воно є інструментом для фіксації рішень, опису функціональності систем, а також відстеження змін [1]. Основні типи документації включають:

- технічні специфікації;
- проєктні плани;
- звіти про тестування;
- користувацькі інструкції.

Якісна документація дозволяє всім учасникам проєкту чітко розуміти вимоги, завдання, обов'язки, етапи виконання завдань й очікувані результати. Це також важливо для майбутньої підтримки та розвитку системи, коли нові учасники або сторонні особи отримують доступ до проєкту. Роль документації в управлінні ІТ-проєктами представлена у таблиці 1.1.

Таблиця 1.1 – Документація в управлінні ІТ-проєктами [2]

Тип документації	Опис	Роль в проєкті
Вимоги користувачів (SRS)	Документ, що описує функціональні та нефункціональні вимоги до системи	Служить основою для проєктування, розробки та тестування
Технічні специфікації (DDS)	Опис архітектури системи, API, протоколів, інтеграцій	Визначає технічні вимоги та стандарти
Проєктний план	Графік виконання завдань, часові рамки, відповідальні особи	Координація етапів проєкту, контроль виконання
Звіти про тестування	Результати тестування компонентів і системи в цілому	Забезпечує якість системи, виявлення помилок
Користувацькі інструкції	Опис функціональних можливостей для кінцевих користувачів	Полегшує використання системи, зменшує навантаження на підтримку

Зауважимо, що вимоги користувачів (SRS) мають важливе значення, оскільки саме на основі цього документа команда розробки будує весь процес створення продукту. SRS дозволяють чітко зрозуміти потреби користувачів і гарантувати, що продукт відповідає очікуванням замовника.

1.1.2 Види документації в IT-проєктах

В IT-проєктах існує кілька видів документації, які супроводжують кожен етап життєвого циклу інформаційних систем (SDLC), щоб забезпечити ефективне планування, розробку, впровадження і підтримку IT-проєкту [3]:

- документація на етапі ініціації та планування проєкту;
- документація на етапі розробки;
- документація на етапі тестування;
- документація на етапі впровадження;
- користувацька документація;
- документація на етапі підтримки.

На етапі ініціації та планування формуються два основних документа – бізнес-вимоги (Business Requirements Document, BRD), що містять опис цілей проєкту, вимог замовників і кінцевих користувачів та технічне завдання (ТЗ) або специфікація вимог (System Requirements Specification, SRS) детально визначає функціональні та нефункціональні вимоги до системи. Є основою для розробки інформаційної системи і часто розробляється під час аналізу вимог. Обидва документи створюються на початкових етапах життєвого циклу IT-проєкту – під час збору вимог і планування.

На етапі розробки створюється архітектурна документація, що описує структуру системи, ключові компоненти, взаємодію між ними та їхній розвиток, і включає різні види UML-діаграм, схеми даних тощо, а також технічна специфікація, що містить деталізований опис кожного елементу системи, інтерфейсів, алгоритмів та інших технічних аспектів, важливих для розробки. Ці документи створюються на етапі проєктування та початку розробки системи.

На етапі тестування створюються тестові сценарії (Test Cases) та плани тестування (Test Plans), що описують підхід до тестування системи, включаючи

список функціональних і нефункціональних вимог, які необхідно перевірити, а також звіти про тестування, що містять результати виконання тестів, фіксують знайдені помилки та статус кожного етапу тестування. Ці документи використовуються на етапі тестування системи, після завершення основної частини розробки.

На етапі впровадження створюються інструкції з налаштування та впровадження (Deployment Guide), які описують кроки, необхідні для встановлення і налаштування системи у робочому середовищі, а також документація для адміністраторів IT-системи, що містить інструкції з управління системою, її моніторингу, відновлення після збоїв та налаштування. Ці документи використовуються на етапі впровадження та інтеграції системи в робоче середовище.

Важливим типом документації є керівництва користувача (User Manuals), які містять інструкції для кінцевих користувачів щодо використання системи, її функцій та інтерфейсів, та онлайн-документація або контекстна довідка (Online Help) – інтегровані в систему допоміжні матеріали, які пояснюють, як працювати з конкретними функціями. Користувацька документація готується на етапі впровадження та підтримки, щоб забезпечити користувачам повноцінний доступ до інформації про роботу з системою [4].

Документація з технічної підтримки (Support Documentation) описує процедури вирішення проблєм, методи відновлення системи у разі збою, а також містить контактну інформацію технічної підтримки. Журнал змін (Change Log) фіксує всі зміни, зроблені в системі під час її підтримки та оновлень. Ці документи необхідні для ефективного управління системою після її впровадження, на етапі супроводу та підтримки [5, 6].

Таким чином, кожен етап життєвого циклу інформаційних систем супроводжується певним типом документації, що забезпечує належний контроль, розвиток та підтримку системи на всіх її етапах [7]. Узагальнення відомостей про види документації в IT-проєктах з прив'язкою до етапів життєвого циклу інформаційних систем (SDLC) представлене в таблиці 1.2.

Таблиця 1.2 – Види документації в IT-проєктах

Види документації	Призначення документації	Етапи SDLC
Бізнес-вимоги (BRD)	Опис цілей проєкту, вимог замовників і кінцевих користувачів.	Етап ініціації та планування
Технічне завдання (SRS)	Специфікація функціональних та нефункціональних вимог до системи.	Етап ініціації та планування
Архітектурна документація	Описує структуру системи, ключові компоненти, взаємодію між ними. Включає UML-діаграми та схеми.	Етап проєктування та розробки
Технічна специфікація (DDS)	Детальний опис елементів системи, інтерфейсів, алгоритмів та технічних аспектів.	Етап проєктування та розробки
Тестові сценарії та плани тестування	Опис підходів до тестування системи, включаючи функціональні та нефункціональні вимоги.	Етап тестування
Звіти про тестування	Фіксація результатів тестування, виявлених помилок та статусу тестів.	Етап тестування
Інструкції з налаштування та впровадження	Кроки для встановлення та налаштування системи у продуктивному середовищі.	Етап впровадження
Документація для адміністраторів	Інструкції з управління системою, моніторингу та відновлення після збоїв.	Етап впровадження та інтеграції
Керівництва користувача (User Manuals)	Інструкції для кінцевих користувачів щодо використання системи та її функцій.	Етап впровадження та підтримки
Онлайн-документація (Online help)	Інтегровані в систему матеріали для користувачів щодо використання конкретних функцій.	Етап впровадження та підтримки
Документація з технічної підтримки	Процедури вирішення проблєм, відновлення системи та контактна інформація підтримки.	Етап підтримки
Журнал змін (Change log)	Фіксація змін, зроблених в системі під час її підтримки та оновлень.	Етап підтримки

До переліку найбільш важливих документів, що мають значний вплив на успіх та ефективність IT-проєктів, належать [7]:

- технічне завдання (ТЗ) або специфікація вимог (SRS);
- архітектурна документація;
- тестова документація;
- користувацька документація (User Manuals);
- журнал змін (Change Log).

Технічне завдання (ТЗ) або специфікація вимог (SRS) є основою для всієї розробки, оскільки визначає функціональні та нефункціональні вимоги до системи. Він використовується на етапі планування і є ключовим для розуміння очікувань замовника та команди розробників. У великих проєктах неправильне

або неповне формулювання вимог може призвести до значних проблєм у майбутньому, таких як невідповідність очікувань або затримки у виконанні проєкту.

Архітектурна документація включає діаграми та описи, які показують структуру системи, взаємодію між компонентами та ключові архітектурні рішення. Цей вид документації важливий для розробників, щоб чітко розуміти, як система функціонує і які компоненти є критичними. При зміні команди розробників або масштабуванні системи, наявність детальної архітектурної документації допомагає швидко адаптувати нових учасників і зберегти єдиний підхід до розвитку.

Тестова документація (плани тестування, тестові сценарії) забезпечує структуру для перевірки якості та надійності системи. Важливою є наявність тестових сценаріїв, які дозволяють виявити помилки ще на ранніх етапах розробки. У системах з високими вимогами до якості та безпеки, як, наприклад, фінансові або медичні системи, якісна тестова документація є надзвичайно важливою для виявлення вразливостей у системі.

Користувацька документація (User Manuals) – основний інструмент для кінцевих користувачів, який допомагає їм успішно працювати з системою. Керівництва користувача мають бути зрозумілими та доступними, особливо для не технічних користувачів. У складних системах, таких як ERP-системи, наявність детальної документації користувача є вирішальним фактором для швидкого впровадження і навчання користувачів.

Журнал змін (Change Log) фіксує всі зміни, внесені в систему протягом її життєвого циклу. Це дозволяє відстежувати внесені модифікації, оцінювати їх вплив на систему і забезпечувати зворотну сумісність. Журнал змін є необхідним для довготривалих проєктів, особливо коли система постійно вдосконалюється або підтримується. Він допомагає підтримувати ясність стосовно того, які зміни були внесені і з яких причин [8-10].

Відомості про важливість окремих видів документації в IT-проєктах узагальнені у таблиці 1.3.

Таблиця 1.3 – Важливість окремих видів документації в ІТ-проєктах

Тип документації	Опис	Етап життєвого циклу	Важливість
Технічне завдання (SRS)	Опис функціональних та нефункціональних вимог до системи.	Ініціація та планування	Неправильне формулювання вимог може призвести до невідповідності очікувань замовника та затримок у виконанні проєкту.
Архітектурна документація	Діаграми і схеми, які описують структуру системи та взаємодію її компонентів.	Проєктування	Важлива при зміні команди або масштабуванні проєкту, забезпечує зрозумілість архітектурних рішень.
Тестова документація	Плани тестування та тестові сценарії для перевірки функціональності та якості системи.	Тестування	Необхідна для виявлення помилок і зниження ризиків у системах з високими вимогами до якості, наприклад, у фінансових або медичних проєктах.
Користувацька документація	Інструкції для кінцевих користувачів щодо використання системи.	Впровадження та підтримка	Важлива для навчання користувачів у складних системах, таких як ERP, що полегшує їх впровадження.
Журнал змін (Change Log)	Описує всі зміни в системі, внесені протягом її життєвого циклу.	Підтримка та супровід	Допомагає відстежувати зміни, оцінювати їх вплив і забезпечує зворотну сумісність, важлива для довготривалих проєктів.

Ця таблиця підсумовує ключові види документації, їхні функції на різних етапах життєвого циклу ІТ-проєкту та важливість для успішного управління проєктом. Кожен з цих типів документації виконує важливу роль на певних етапах життєвого циклу проєкту та допомагає забезпечити його ефективне планування, розробку, тестування і підтримку.

1.2 Теоретичні основи автоматизації документування

1.2.1 Принципи автоматизації процесів документування

Автоматизація процесів у сфері документування ІТ-проєктів базується на кількох принципах, які покликані підвищити ефективність та точність створення і підтримки документації. Зокрема, дотримання цих принципів дозволяє знизити

ризики людських помилок, прискорити робочі процеси та забезпечити актуальність документів [11].

1. Принцип мінімізації участі людини. Автоматизація спрямована на те, щоб звести до мінімуму ручне втручання. Цей принцип дозволяє зменшити кількість помилок, що виникають під час створення документів, і підвищити швидкість процесу.

2. Принцип автоматичного оновлення документації. Документація має оновлюватися автоматично разом із змінами в коді або інших системах. Дотримання цього принципу забезпечує актуальність і точність інформації.

3. Принцип інтеграції з іншими системами. Автоматизовані системи документування повинні бути інтегровані з іншими частинами IT-інфраструктури, такими як системи контролю версій (Git), інструменти для CI/CD (Jenkins, GitLab CI), та системи управління проєктами (Jira, Trello).

4. Принцип контролю версій. Автоматизація повинна передбачати можливість відстеження всіх змін у документації (контролю версій документації), щоб за потреби мати можливість легко повернутися до попередніх версій або аналізувати зміни.

5. Принцип автоматизованої генерації документації. Документи повинні генеруватись автоматично на основі коду, специфікацій або метаданих проєкту. Це дозволить зекономити час на створення документації, а також забезпечити актуальність документації – її відповідність реальним процесам.

1.2.2 Методології автоматизації документування IT-проєктів

Можна виділити кілька основних «традиційних» підходів до автоматизації документування IT-проєктів [12].

Автоматизація документування в рамках DevOps – документація генерується і оновлюється автоматично під час процесів безперервної інтеграції та доставки (CI/CD). Характерна риса – постійна актуалізація документації разом із змінами в коді. Переваги: постійне оновлення документації; інтеграція з існуючими інструментами розробки. Недоліки: необхідність глибокої інтеграції з DevOps-інструментами. Приклад реалізації: Jenkins + Sphinx.

Автоматизована генерація документації на основі коду. Цей підхід передбачає використання інструментів, таких як Swagger або Sphinx, для автоматичного створення документації на основі коментарів у коді або API. Він дозволяє значно зекономити час на створення технічної документації. Переваги: зменшення людських помилок; швидкість генерації документації. Недоліки: вимагає підтримки структури коду та коментарів. Приклад реалізації: використання Swagger для API-документації.

Візуалізація процесу автоматизації документування через CI/CD та автогенерації документації представлена на діаграмі, що показує процес автоматизації документування під час IT-розробки, де зміни в коді автоматично ініціюють генерацію нової документації (рисунок 1.1).

Рисунок 1.1 – Діаграма процесу автоматизації документування через CI/CD та автогенерації документації

Автоматизація через системи управління знаннями. Використання інструментів для управління знаннями, таких як Confluence або SharePoint, дозволяє автоматизувати створення звітів та документів у великих проєктах з великою кількістю учасників. Основна перевага – можливість централізованого зберігання документів та легкий доступ для всіх членів команди. Переваги: легкий доступ до документів; можливість масштабування. Недоліки: висока вартість налаштування для великих команд. Приклад реалізації: використання Confluence для проєктної документації.

Таким чином, основні сучасні підходи до автоматизації документування базуються на інтеграції з CI/CD процесами, автогенерації документації на основі коду або використанні централізованих систем управління знаннями. Вибір підходу залежить від масштабу проєкту та технічної інфраструктури команди розробників. Крім того, останнім часом з'явились також і нові важливі аспекти досліджень у цьому напрямку [13]. Узагальнення інформації щодо різних підходів у напрямку автоматизації документування ІТ-проєктів представлене у таблиці 1.4.

Таблиця 1.4 – Порівняння різних підходів до автоматизації документування

Підхід	Основні переваги	Недоліки	Приклад
DevOps і CI/CD інтеграція	Постійна актуалізація документації; зменшення ручного втручання	Необхідність інтеграції з інструментами CI/CD	Jenkins + Sphinx
Автогенерація документації на основі коду	Швидке створення технічної документації, інтерактивність	Потребує підтримки коментарів і структури коду	Swagger для API
Управління знаннями через автоматизовані системи	Централізоване зберігання документів, легкий доступ для команди	Висока вартість для великих проєктів	Confluence
Автоматизація документації на основі ІІІ	Штучний інтелект автоматично класифікує та структурує документацію, покращує пошук	Необхідність налаштування алгоритмів, висока складність впровадження	AI-driven системи документування (ML-моделі)
Хмарні технології для автоматизації документації	Доступність документів в будь-який час, висока надійність збереження даних	Вартість хмарних рішень може бути високою	Microsoft Azure, AWS

Різні підходи, представлені у таблиці 1.4, по-різному підвищують ефективність автоматизації документування. Всі вони мають значний потенціал для IT-проєктів, які потребують високого рівня інтеграції та управління даними.

1.2.3 Програмні технології автоматизації документування

Значна кількість сучасних програмних технологій використовуються, щоб автоматизувати процес створення та оновлення документації в IT-проєктах.

Jenkins – потужний інструмент для автоматизації процесів CI/CD, який дозволяє налаштовувати пайплайни, які автоматично збирають документацію на основі оновлень у коді. Це може бути використано для генерації технічної документації на основі коду, який розробляється. Основні можливості Jenkins [14]: автоматичне генерування документації під час збирання проєкту; інтеграція з системами контролю версій, такими як Git [15]; підтримка різних форматів документації (Markdown, HTML).

Sphinx – інструмент для генерації документації, який зазвичай використовується в Python-проєктах. Він дозволяє автоматично генерувати документацію на основі коментарів у коді та підтримує різні формати виводу (HTML, PDF, ePub). Основні можливості Sphinx: підтримка автодокументації на основі коду; підтримка різних мов програмування; можливість інтеграції з Git для контролю версій [16].

Swagger – інструмент для автоматизованої генерації документації для API. Дозволяє розробникам описувати API, а система автоматично генерує документацію, яку можна переглядати через вебінтерфейс. Основні можливості Swagger: генерація інтерактивної документації для REST API; інтеграція з мовами програмування, такими як Java, Python, PHP; автоматичне оновлення документації після змін в API [17].

Doxygen – інструмент для автоматизації документування коду на C++, Java, Python та інших мовах. Дозволяє створювати технічну документацію на основі коментарів у коді. Основні можливості Doxygen: автоматичне генерування документації для великих проєктів; підтримка UML-діаграм для візуалізації структури проєкту; генерація документації в форматі HTML, LaTeX, PDF [18].

GitLab CI/CD – інструмент для безперервної інтеграції та доставки, який дозволяє налаштовувати автоматичні пайплайни для збирання проєкту і документування. Може генерувати документацію на основі специфікацій або результатів виконання тестів. Основні можливості [19]: підтримка автоматизованого генерування документації; інтеграція з GitLab-репозиторіями; можливість створення публічної документації через GitLab Pages [20]. Діаграма на рисунку 1.2 описує автоматизований процес генерування документації на основі коду в рамках CI/CD. Документація автоматично генерується під час збирання проєкту та публікується для подальшого використання.

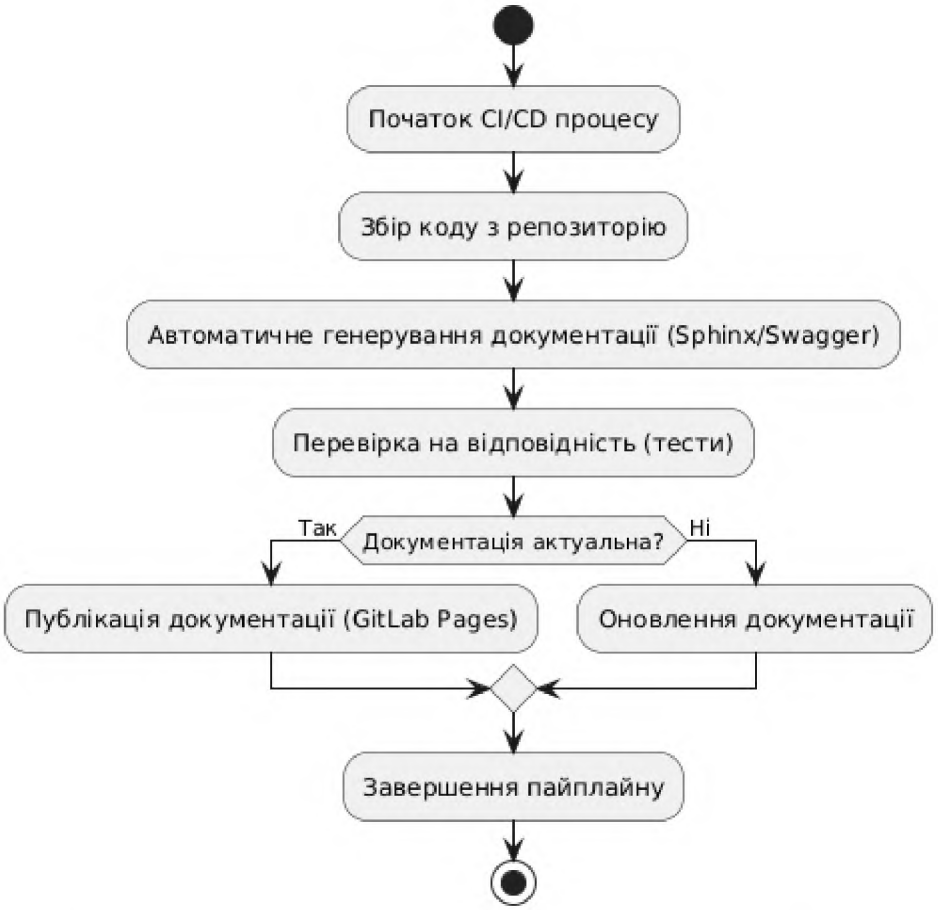

Рисунок 1.2 – Алгоритм автоматизації документування ІТ-проєктів у процесі CI/CD

Порівняння сучасних технологій для автоматизації документування представлене у таблиці 1.5.

Таблиця 1.5 – Порівняння сучасних засобів автоматизації документування

Інструмент	Основні можливості	Переваги	Недоліки
Jenkins	Автоматичне генерування документації під час CI/CD процесів	Гнучкість налаштування, інтеграція з різними мовами програмування	Складність налаштування для нетехнічних користувачів
Sphinx	Генерація документації на основі коментарів у коді	Підтримка різних форматів виводу, легке налаштування для Python	Орієнтований в основному на Python
Swagger	Автогенерація документації для API	Інтерактивна документація, інтеграція з популярними мовами	Вузька спеціалізація на API
Doxygen	Генерація технічної документації з підтримкою UML-діаграм	Підтримка кількох мов програмування, потужний інструмент для великих проєктів	Складність налаштування для невеликих команд
GitLab CI/CD	Автоматизація процесів збірки і документування через пайплайни	Інтеграція з GitLab, можливість публікації документації	Вимагає розгорнутої CI/CD інфраструктури

Таким чином, автоматизація процесу документування IT-проєктів підвищує ефективність командної роботи, знижує кількість помилок, забезпечує актуальність документації на всіх етапах розробки. Сучасні інструменти, такі як Jenkins, Sphinx, Swagger та інші, дозволяють налаштувати гнучкі та потужні автоматизовані процеси для підтримки документації в IT-проєктах.

1.3 Методології та інструменти документування

1.3.1 Сучасні підходи до документування IT-проєктів

Існуючі методології управління IT-проєктами по-різному підходять до питання документування. Вибір підходу залежить від характеру проєкту, розміру команди та індивідуальних потреб організації.

Водоспадна модель (Waterfall) передбачає покрокове виконання етапів проєкту, де кожен етап має бути завершений перед тим, як розпочати наступний. У цьому підході документування є важливою частиною кожної фази проєкту. Специфікації (SRS), архітектурні та технологічні рішення (DDS) та тестова документація розробляються на ранніх стадіях і не змінюються під час виконання

проєкту. Перевагами такого підходу є те, що повна документація створюється ще до початку розробки, також її легко зрозуміти новим членам команди. Основні недоліки в тому, що створена документація швидко застаріває через зміни в проєкті, й важко інтегрувати зміни до документації проєкту на пізніх етапах [21].

Гнучка методологія розробки (Agile) орієнтована на гнучкість та швидкі зміни [22]. Документування в цій методології мінімізується, основна увага приділяється безперервному розвитку продукту – документація підтримується лише в обсязі, необхідному для поточних потреб, постійно оновлюється. Основні переваги: документація проєкту є актуальною на кожному етапі, є можливість швидко реагувати на зміни у проєкті, і відповідно оновлювати документацію. Недоліками є брак детальної документації, що може становити проблєми, а також через нестачу детальної документації важко інтегрувати нових членів команди.

Порівняння основних підходів до документування ІТ-проєктів представлене у таблиці 1.6.

Таблиця 1.6 – Порівняння підходів до документування проєктів у різних методологіях управління ІТ-проєктами

Методологія	Основні характеристики	Переваги	Недоліки
Waterfall	Документація створюється на початку, жорстка послідовність етапів	Повна документація до початку розробки	Складність оновлення документації при змінах
Agile	Гнучкий підхід, мінімум документації, акцент на продукт	Швидке реагування на зміни, актуальна документація	Можливий брак детальної документації
DevOps	Інтеграція розробки та експлуатації, автоматизація документації	Постійне оновлення документації, автоматизація збору	Високі технічні вимоги, необхідність інтеграції інструментів

Методологія DevOps поєднує розробку та операційне забезпечення, що забезпечує безперервність потоку – постійний розвиток, тестування та оновлення системи [23]. Документування тут також є безперервним процесом, тому його автоматизація є важливою. Основна увага приділяється автоматизації процесу збору документації та його інтеграції з CI/CD-процесами. Перевагами є автоматизоване створення документації та постійне оновлення документації

разом із змінами в коді. Основні недоліки: необхідність інтеграції інструментів автоматизації документування, високі вимоги до технічної компетенції команди.

1.3.2 Платформи для автоматизованого документування

Для забезпечення ефективної роботи з документацією в IT-проєктах використовуються різні інструменти, кожен з яких має свої особливості та підходить для різних задач [24].

Confluence – корпоративна вікісистема для створення, зберігання та спільної роботи з документацією, яка підтримує інтеграцію з Jira, що дозволяє пов'язати проєктні завдання з відповідною документацією. Основні можливості Confluence: створення структурованих документів; спільна робота в реальному часі; інтеграція з інструментами для управління проєктами [25].

SharePoint – платформа від Microsoft для управління документами та спільної роботи. Вона дозволяє створювати бібліотеки документів, налаштовувати робочі процеси та надавати доступ до документації для великої кількості користувачів. Основні можливості SharePoint: централізоване зберігання документів; розширені функції керування доступом; інтеграція з іншими продуктами Microsoft [26].

Порівняння популярних платформ для документування IT-проєктів представлене у таблиці 1.7 [27-29].

Таблиця 1.7 – Порівняння платформ для документування IT-проєктів

Інструмент	Основні можливості	Переваги	Недоліки
Confluence	Створення, зберігання, спільна робота, інтеграція з Jira	Простота використання, інтеграція з інструментами управління проєктами	Висока вартість для великих команд
SharePoint	Бібліотеки документів, налаштування робочих процесів, інтеграція з Microsoft	Централізоване зберігання, налаштування доступів	Складність налаштування для нетехнічних користувачів
GitBook	Контроль версій, підтримка Markdown, інтеграція з репозиторіями Git	Простий для розробників, гнучкість	Обмежені можливості для нетехнічних команд

GitBook – інструмент для створення технічної документації, орієнтований на розробників програмного забезпечення. Він дозволяє легко інтегрувати

документацію з репозиторіями коду, підтримує формат Markdown і надає можливість контролю версій документації. Основні можливості GitBook: підтримка контролю версій через Git, інтеграції з популярними хостингами git-репозиторіїв, простий у використанні редактор Markdown, автоматизоване оновлення документації в реальному часі [30-34].

Діаграма на рисунку 1.3 описує процес автоматизованого документування за методологією DevOps з використанням GitBook. Ця діаграма показує, як автоматизовані інструменти для документування інтегруються в процес розробки і підтримки коду, що забезпечує постійне оновлення документації разом з оновленнями в коді.

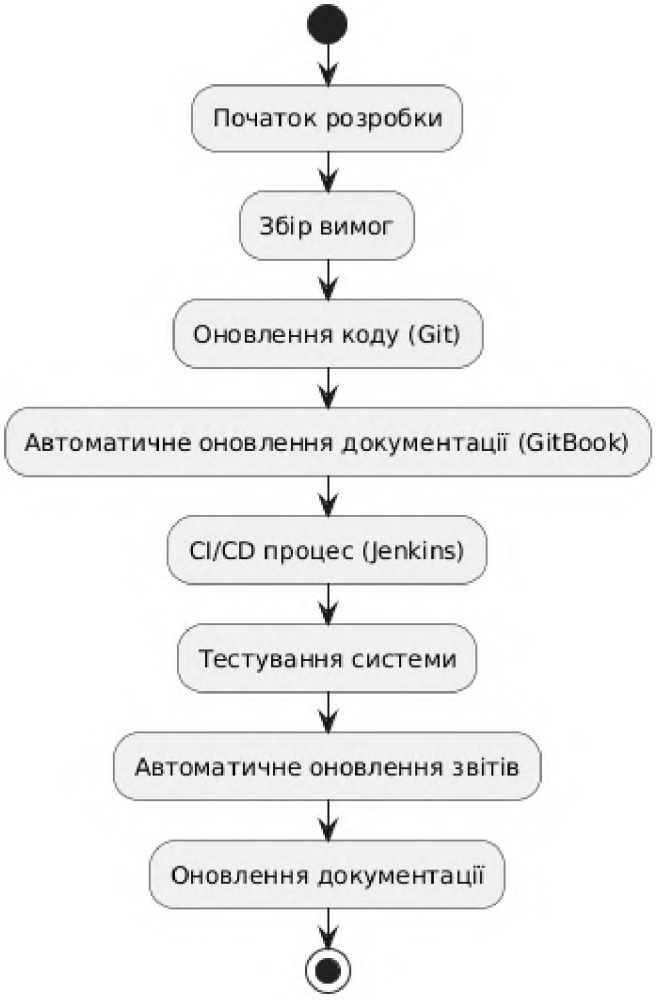

Рисунок 1.3 – Процес автоматизованого документування за методологією DevOps з використанням GitBook

Таким чином, вибір підходу до документування та відповідних інструментів залежить від вимог проєкту, масштабу команди та рівня автоматизації процесів. Автоматизація допомагає знизити ризики та забезпечити актуальність документації на кожному етапі життєвого циклу IT-проєкту.

1.3.3 Безкоштовні рішення для автоматизованого документування

У сучасних IT-проєктах важливо використовувати інструменти, які забезпечують легке створення, оновлення та зберігання документації. Окрім платних рішень, існує багато безкоштовних інструментів, що забезпечують достатній функціонал для роботи з документацією, особливо для невеликих команд або проєктів із обмеженим бюджетом. Нижче наведено огляд популярних безкоштовних інструментів [35]:

GitBook (безкоштовний план з обмеженим функціоналом) – орієнтований на розробників і дозволяє інтегрувати документацію з репозиторіями коду. Він використовує Markdown для написання текстів і підтримує контроль версій через Git. Вважається найкращим рішенням для технічної документації у проєктах з відкритим кодом. Основні можливості: підтримка Markdown забезпечує легке редагування; інтеграція з Git для контролю версій; публікація документації в реальному часі. Переваги: простота використання для технічних команд; легке налаштування та інтеграція з Git. Недоліки: недостатній функціонал безкоштовної версії для великих команд або складних проєктів [36-38].

Google Docs (повністю безкоштовний) – один із найбільш популярних інструментів для створення документації завдяки простоті використання та підтримці спільної роботи в режимі реального часу. Підходить для невеликих і середніх команд, як безкоштовний інструмент для роботи з текстовими документами. Основні можливості: спільне редагування документів у реальному часі; автоматичне збереження версій документа; підтримка інтеграції з Google Drive для зберігання та організації файлів. Переваги: повністю безкоштовний, доступний для всіх користувачів; легкий доступ з будь-якого пристрою через веббраузер. Недоліки: відсутність спеціалізованих функцій для IT-проєктів (наприклад, немає інтеграції з системами контролю версій) [39-47].

GitHub Pages (безкоштовний) – популярний інструмент для технічної документації, написаної у форматі Markdown. Дозволяє розміщати документацію безпосередньо з репозиторіїв на GitHub. Забезпечує автоматичну публікацію на основі контенту, який зберігається у репозиторії. Основні можливості: використання Markdown для створення документації; автоматичний хостинг документації через GitHub Pages; підтримка контролю версій через Git. Переваги: повністю безкоштовний для відкритих проєктів; проста інтеграція з існуючими проєктами на GitHub. Недоліки: відсутність інтеграцій для бізнес-інструментів, таких як Jira або Confluence [48-50].

Docusaurus (безкоштовний, open-source) – фреймворк для створення вебсайтів документації, який використовує React [51]. Його можна інтегрувати з існуючими інструментами розробки та репозиторіями коду, що є зручним для великих IT-проєктів. Підтримує багатомовність, має зручний інтерфейс для навігації по документації. Основні можливості: використання Markdown або React для створення документації; підтримка багатомовності та пошуку по документах4 можливість інтеграції з процесами CI/CD. Переваги: безкоштовний та з відкритим кодом; підходить для великих проєктів з великою кількістю документації. Недоліки: вимагає певних знань з програмування для налаштування та підтримки.

Notion (безкоштовний план з обмеженим функціоналом) – багатофункціональний інструмент для управління проєктами та створення документації. Підтримує таблиці, списки, вбудовані файли та документи, зручний для невеликих команд. Хоча його безкоштовний план має обмеження на кількість сторінок, він все одно підходить для багатьох завдань. Основні можливості: створення структурованої документації з використанням шаблонів; підтримка спільної роботи та управління завданнями; інтеграція з іншими інструментами через API. Переваги: інтуїтивно зрозумілий інтерфейс, можливість роботи в режимі реального часу; підходить для управління як документацією, так і проєктами. Недоліки: обмежений функціонал безкоштовної версії для великих команд; відсутність технічних інтеграцій для CI/CD або Git [52-55].

У таблиці 1.8 представлене порівняння безкоштовних інструментів для документування проєктів.

Таблиця 1.8 – Порівняння безкоштовних інструментів для документування проєктів

Інструмент	Основні можливості	Переваги	Недоліки
GitBook	Markdown, інтеграція з Git, публікація в реальному часі	Легке використання, інтеграція з Git	Обмежений функціонал безкоштовної версії
Google Docs	Спільна робота, автоматичне збереження, інтеграція з Google Drive	Повністю безкоштовний, зручний для командної роботи	Немає спеціалізованих функцій для IT-проєктів
GitHub Pages	Markdown, автоматична публікація, інтеграція з Git	Безкоштовний, інтеграція з Git	Обмежена функціональність для бізнес-проєктів
Docusaurus	Markdown, підтримка багатомовності, інтеграція з CI/CD	Відкритий код, підходить для великих проєктів	Вимагає знань програмування для налаштування
Notion	Створення структурованої документації, управління завданнями	Інтуїтивний інтерфейс, підтримка багатофункціональних документів	Обмежений функціонал безкоштовної версії, відсутність технічних інтеграцій

Узагальнюючи таблицю 1.8, можна відзначити, що кожен безкоштовний інструмент для документування проєктів має свої особливості та обмеження. GitBook і GitHub Pages відзначаються інтеграцією з Git, що забезпечує актуальність документації, однак мають обмежену функціональність у безкоштовних версіях. Google Docs є зручним для командної роботи завдяки простоті та доступності, але не містить спеціалізованих функцій для IT-проєктів. Docusaurus підходить для великих проєктів із підтримкою багатомовності, однак потребує технічних навичок для налаштування. Notion пропонує гнучкий інтерфейс для структурованої документації, але обмежується у функціональних інтеграціях. Вибір інструменту залежить від специфіки проєкту та потреб команди.

Процес автоматизованого документування з використанням GitHub Pages і Markdown описує діаграма на рисунку 1.4.

Рисунок 1.4 – Процес автоматизованого документування з використанням GitHub Pages і Markdown

Ця діаграма демонструє автоматизований процес створення та оновлення документації за допомогою GitHub Pages і Markdown, що вважається зручним для невеликих команд або відкритих проєктів.

Таким чином, безкоштовні інструменти можуть бути ефективними рішеннями для автоматизації документування в IT-проєктах, особливо для малих команд і проєктів з обмеженими бюджетами.

Висновки до розділу 1

У першому розділі розглянуто теоретичні та методологічні основи автоматизації документування в IT-проєктах, а також інструменти та методології для ефективного створення й підтримки документації.

Встановлено, що документування відіграє важливу роль в управлінні проєктами, забезпечуючи прозорість процесів, фіксацію вимог та координацію дій між командами. Визначено основні види документації, які супроводжують кожен етап життєвого циклу: бізнес-вимоги, технічні специфікації, архітектурна документація, тестові плани, інструкції, керівництва користувачів і технічна підтримка.

Аналіз показав, що ручне документування є неефективним через ймовірність помилок, застарівання інформації та витрати часу. Автоматизація вирішує ці проблєми завдяки інтеграції сучасних інструментів, принципів CI/CD, генерації документації з вихідного коду або API-специфікацій та автоматичному оновленню.

Проаналізовано методології, зокрема Agile і DevOps, та інструменти GitLab CI/CD, Jenkins, Swagger, Sphinx і MkDocs, які дозволяють автоматизувати створення та публікацію документації, мінімізуючи ручну працю.

Особливу увагу приділено безкоштовним рішенням, ефективним для малих і середніх проєктів. MkDocs, Swagger і Docusaurus демонструють високу продуктивність при мінімальних витратах, що робить їх популярними серед команд з обмеженими ресурсами.

РОЗДІЛ 2
АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ТЕНДЕНЦІЙ В АВТОМАТИЗАЦІЇ ДОКУМЕНТУВАННЯ ІТ-ПРОЄКТІВ

2.1 Аналіз існуючих систем та підходів автоматизації документування

2.1.1 Проблєми ручного документування

Зважаючи на велику кількість та різноманіття документації, ручне документування в ІТ-проєктах має низку проблєм, що ускладнюють підтримку високої якості та актуальності документації [56-61]:

- велике навантаження на команду проєкту – створення документації вручну займає багато часу та ресурсів, відволікає фахівців від основної роботи над проєктом;
- людські помилки – через фактор людського впливу можливі помилки, пропуски або невідповідність документації реальним змінам у проєкті;
- низька актуальність документації – оновлення документації вручну часто затримується, що призводить до її невідповідності реальному стану проєкту;
- неможливість швидкого пошуку та аналізу – ручне документування не дозволяє ефективно обробляти великі обсяги даних, що ускладнює пошук потрібної інформації.

Основні проблєми ручного документування узагальнені у таблиці 2.1.

Таблиця 2.1 – Проблєми ручного документування

Проблєма	Опис	Наслідки
Велике навантаження	Надмірні витрати часу і ресурсів на створення та підтримку документації вручну	Зниження ефективності роботи команди
Людські помилки	Можливість пропуску важливої інформації або внесення некоректних даних	Невідповідність документації реальним процесам
Низька актуальність	Відставання оновлення документації від змін у проєкті	Збільшення ризиків через невірну інформацію
Ускладнений пошук	Неможливість швидкого пошуку і обробки великих обсягів документів	Затримки у прийнятті рішень

Процес документування в ІТ-проєкті та проблєми, пов'язані з ручним документуванням схематично представлені на діаграмі діяльності. Ця діаграма показує, що ручне документування потребує постійних доопрацювань і часто призводить до затримок через необхідність повторного збору вимог або корекції інформації (рисунок 2.1).

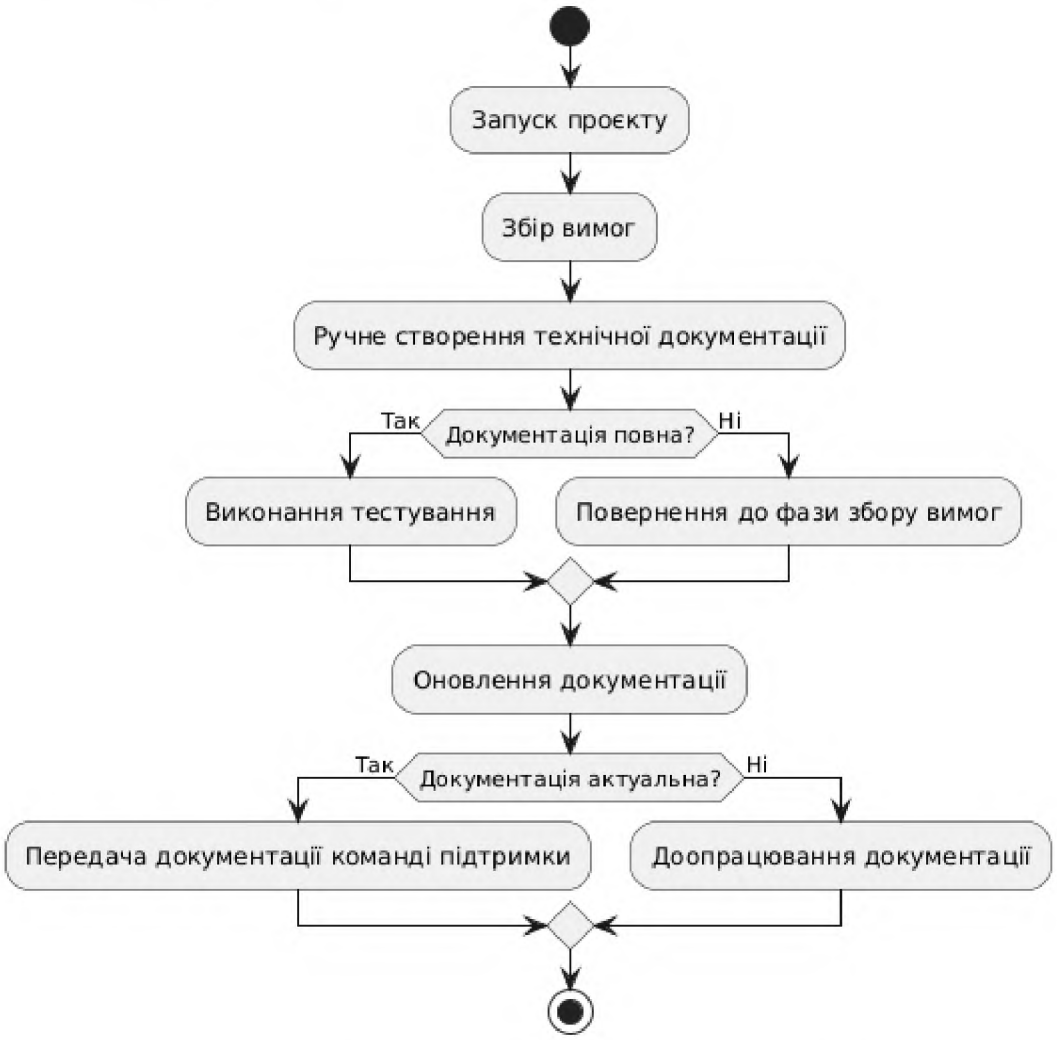

Рисунок 2.1 – Процес документування в ІТ-проєкті та проблєми, пов'язані з ручним документуванням

Сучасним рішенням цих проблєм є автоматизація документування, яка дозволяє швидко та безперебійно оновлювати документацію ІТ-проєкту, синхронізуючи її зі змінами у вихідному коді або специфікаціях. Це значно зменшує ризик помилок, спричинених людським фактором, забезпечує актуальність даних на всіх етапах життєвого циклу системи та полегшує доступ

до необхідної інформації для всієї команди. Як результат, автоматизація сприяє підвищенню продуктивності команди, оптимізації робочих процесів та зниженню витрат на підтримку й оновлення документації, що є критично важливим для успішного виконання складних і динамічних IT-проєктів [62, 63].

2.1.2 Приклади автоматизації документування у реальних проєктах

Автоматизація документування відіграє важливу роль у багатьох сучасних IT-проєктах, де існує потреба зменшити ручну працю і забезпечити своєчасне оновлення документації. Нижче наведено кілька прикладів використання автоматизації документування в реальних проєктах.

GitLab CI/CD є прикладом інтегрованого рішення, яке дозволяє автоматизувати процес створення та оновлення документації. Проєкти, які використовують GitLab, автоматично генерують технічну документацію під час виконання процесів CI (Continuous Integration) або CD (Continuous Deployment). GitLab CI/CD дозволяє відстежувати всі зміни у коді та відповідно оновлювати документацію в реальному часі. Наприклад, автоматичне генерування документації API за допомогою Swagger у пайплайнах CI/CD дозволяє постійно підтримувати актуальний стан документації без необхідності її ручного оновлення.

Jenkins – популярний інструмент, який дозволяє автоматизувати процес документування. У великих IT-проєктах використовується для запуску скриптів, які автоматично генерують документацію після успішного збирання коду або виконання тестів. Наприклад, автоматичне генерування документації за допомогою Sphinx дозволяє оновлювати технічну документацію після кожного циклу збирання та тестування коду. Це зменшує кількість помилок у документації та підтримує її актуальною.

Використання хмарних рішень, таких як Microsoft Azure або AWS, також дозволяє забезпечити безперервний доступ до документації і автоматично її оновлювати. У багатьох великих проєктах хмарні платформи інтегруються з інструментами для автоматизації документації, що дозволяє значно підвищити ефективність процесу документування. Наприклад, використання AWS

CodePipeline для інтеграції з Sphinx і автоматичного створення документації для вебдодатків дозволяє постійно зберігати документацію в актуальному стані та надавати доступ до неї всім членам команди з будь-якого місця.

Приклади використання автоматизації документування в IT-проєктах узагальнені у таблиці 2.2.

Таблиця 2.2 – Приклади використання автоматизації документування в IT-проєктах

Інструменти	Опис	Результати
GitLab CI/CD, Swagger	Автоматичне генерування документації API під час CI/CD процесу	Постійно актуальна документація без ручного втручання
Jenkins, Sphinx	Автоматизація генерування документації після тестування	Зниження кількості помилок у документації, автоматизація на кожному циклі
AWS CodePipeline, Sphinx	Хмарна автоматизація документування для вебдодатків	Безперервний доступ до актуальної документації для всієї команди з будь-якого місця

Автоматизація документування за допомогою сучасних інструментів, таких як GitLab CI/CD, Swagger, Jenkins та AWS CodePipeline, забезпечує високу ефективність процесу створення та підтримки документації в IT-проєктах. Як показано у таблиці 2.2, інтеграція цих рішень дозволяє мінімізувати ручну працю, підвищити точність документації та гарантувати її актуальність у реальному часі. Автоматизація є особливо корисною для команд, які працюють у динамічному середовищі, де зміни в коді та вимогах відбуваються постійно. Впровадження цих інструментів не лише покращує якість документації, а й сприяє оптимізації робочих процесів та забезпечує безперервний доступ до необхідної інформації для всієї команди.

2.1.3 Огляд поточних рішень автоматизації документування

Сучасний IT-ринок пропонує різноманітні рішення для автоматизації документування, які допомагають зменшити час на ручну працю і забезпечити точність та актуальність документації. Дослідження цього ринку виявило кілька популярних інструментів, які широко використовуються в різних IT-проєктах.

GitBook – хмарна платформа для створення та автоматизації технічної документації. Підтримує інтеграцію з репозиторіями Git, що дозволяє автоматично публікувати і оновлювати документацію. GitBook часто використовується для публікації технічних специфікацій та керівництв.

Swagger – популярний інструмент для автоматизованого документування API. Дозволяє розробникам автоматично генерувати документацію на основі специфікацій API, що забезпечує актуальність та інтерактивність документації.

Confluence від Atlassian – потужна платформа для управління документами, яка часто використовується для централізованого зберігання та спільної роботи з документацією в середніх та великих командах розробників. Confluence дозволяє автоматизувати створення звітів за рахунок інтеграції з іншими продуктами Atlassian, такими як Jira.

Sphinx – генератор документації, який широко використовується для автоматизації створення технічної документації на основі коду. Він підтримує різні мови програмування і також може інтегруватися з CI/CD пайплайнами.

Характеристики поточних рішень для автоматизації документування представлені у таблиці 2.3.

Таблиця 2.3 – Характеристики поточних рішень для автоматизації документування

Інструмент	Основні можливості	Переваги	Недоліки
GitBook	Хмарна платформа для створення та публікації документації	Інтуїтивний інтерфейс, інтеграція з Git	Обмежений функціонал у безкоштовній версії
Swagger	Генерація документації для API на основі специфікацій	Автоматична документація для API, інтерактивність	Обмежений до API
Confluence	Платформа для спільної роботи з документами	Централізоване зберігання, інтеграція з Jira	Висока вартість для великих команд
Sphinx	Автоматизація створення документації на основі коду	Підтримка різних форматів виводу, легка інтеграція з CI/CD	Потребує знань технічних деталей для налаштування

Діаграма на рисунку 2.2 описує процес автоматизованого створення документації з використанням CI/CD інструментів, демонструє, як система

автоматично створює і публікує документацію після кожної зміни в коді, що допомагає підтримувати її актуальною і точною на кожному етапі життєвого циклу проєкту.

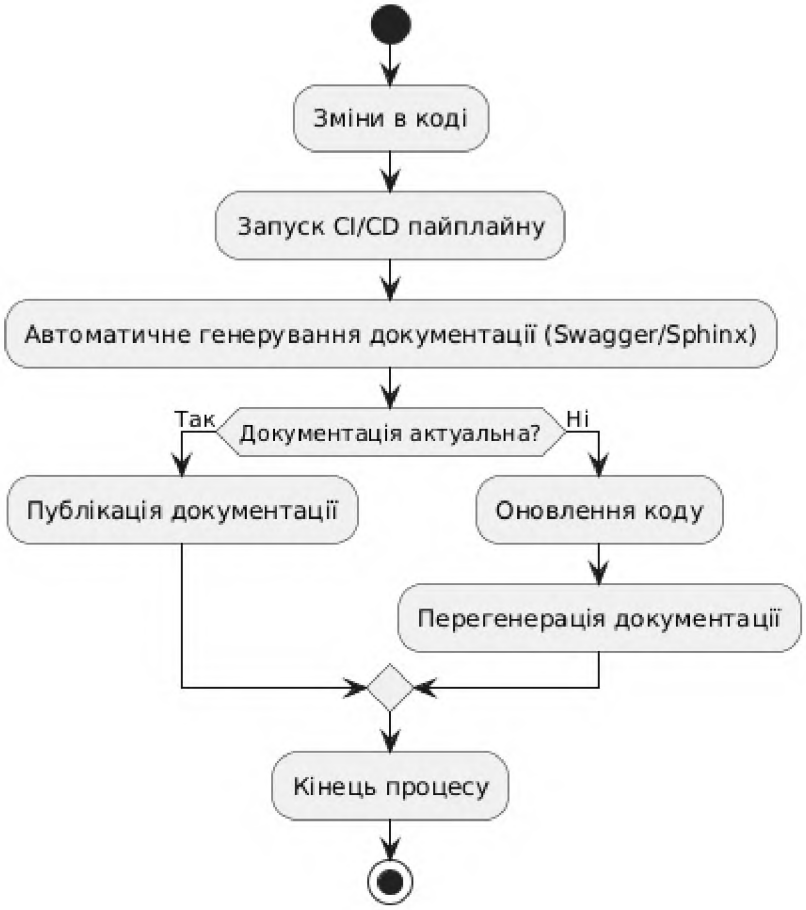

Рисунок 2.2 – Діаграма процесу автоматизованого створення документації з використанням інструментів CI/CD

Незважаючи на різноманіття існуючих засобів, сучасний ринок програмних рішень для автоматизації документування активно розвивається, постійно з'являються нові інструменти, які дозволяють значно підвищити ефективність роботи з документацією. Деякі з цих інструментів є частиною великих екосистем, що дозволяє інтегрувати їх з іншими інструментами розробки та управління проєктами. Прикладами таких систем є інструменти Redoc [64], MkDocs [65], Read the Docs [66], Docusaurus [67], Typora [68] та ін.

Redoc є одним із найпопулярніших інструментів для створення документації API на основі специфікацій OpenAPI. Має інтуїтивний інтерфейс для перегляду документації, який легко інтегрується з іншими системами. Основні можливості Redoc: автоматичне генерування документації на основі специфікацій OpenAPI, підтримка великої кількості форматів та інструментів для публікації та інтерактивна документація для API з прикладами запитів. Головними перевагами Redoc є простота інтеграції з OpenAPI, гнучкість у налаштуваннях інтерфейсу.

MkDocs – інструмент для створення статичних сайтів документації, що підтримує Markdown. Популярний завдяки простоті у використанні та інтеграції з репозиторіями Git для автоматичного оновлення документації. Основні можливості MkDocs: підтримка Markdown для написання документації, автоматичне збирання та оновлення документації при змінах у репозиторії, просте налаштування тем для створення естетично привабливих сайтів документації. Переваги MkDocs: легкість налаштування та використання для невеликих проєктів, інтеграція з Git, що дозволяє автоматизувати оновлення документації.

Read the Docs – платформа для хостингу документації, яка автоматично будує та публікує документацію з репозиторіїв Git. Вона підтримує зовнішні інструменти для автоматизації створення документації, такі як Sphinx. Основні можливості платформи Read the Docs: автоматичне створення документації з репозиторіїв Git, підтримка Sphinx та Markdown, хостинг документації з можливістю автоматичного оновлення. Переваги Read the Docs: безкоштовний хостинг для документації з відкритим вихідним кодом, автоматичне оновлення документації без додаткових зусиль з боку розробників.

Docusaurus – інструмент для створення документації на основі React, зручний для великих проєктів з багатомовною підтримкою та інтеграцією з існуючими інструментами розробки. Основні можливості Docusaurus: генерація документації на основі React, підтримка багатомовності, інтеграція з системами контролю версій та CI/CD. Переваги Docusaurus: гнучкість (завдяки використанню React), можливість створення складних інтерфейсів для документації.

Typora – інструмент для редагування Markdown, який дозволяє створювати просту технічну документацію. Використовується для невеликих проєктів або команд, що працюють з Markdown. Основні можливості Typora: підтримка Markdown для створення документації, прямий вивід у різні формати (включаючи PDF і HTML), інтуїтивний інтерфейс для редагування. Переваги Typora: легкість у використанні для невеликих проєктів, відсутність потреби у додаткових налаштуваннях для генерації документації.

Порівняння новітніх рішень для автоматизації документування представлене у таблиці 2.4.

Таблиця 2.4 – Порівняння нових рішень для автоматизації документування

Інструмент	Основні можливості	Переваги	Недоліки
Redoc	Генерація документації API на основі OpenAPI	Інтерактивна документація, легка інтеграція з OpenAPI	Обмежений до використання для API
MkDocs	Створення статичних сайтів документації на основі Markdown	Простота використання, інтеграція з Git	Не підходить для складних проєктів з великою кількістю функціоналу
Read the Docs	Автоматичне створення та хостинг документації	Безкоштовний хостинг для відкритих проєктів, автоматичне оновлення	Вимагає певних знань налаштування для використання Sphinx
Docusaurus	Створення документації на основі React	Підтримка багатомовності, гнучкість завдяки React	Складність налаштування для нетехнічних користувачів
Typora	Просте редагування Markdown та експорт у різні формати	Легкість у використанні, швидка генерація документації	Не підходить для великих проєктів або інтеграції з CI/CD

Таким чином, сучасний IT-ринок програмного забезпечення пропонує широкий спектр рішень для автоматизації документування, кожне з яких орієнтоване на різні потреби проєктів і команди. Для малих проєктів ефективними є легкі та безкоштовні інструменти, такі як MkDocs або Swagger, які забезпечують просту інтеграцію з системами контролю версій та мінімальні вимоги до налаштування. У середніх та великих проєктах, де обсяги документації значні й потрібна підтримка складних процесів, перевагу надають таким інструментам, як Jenkins, GitLab CI/CD та Confluence, які дозволяють автоматизувати весь цикл документування – від генерування до публікації й підтримки актуальності.

Вибір конкретного рішення залежить від масштабу проєкту, гнучкості інструменту для інтеграції у вже існуючі процеси розробки та рівня технічної підготовки команди. Велике значення мають й особливості робочого процесу: для проєктів, орієнтованих на API, пріоритетним буде використання Swagger або Redoc, тоді як для систем із розгалуженою архітектурою й потребою у багатосторінковій документації більше підійде Sphinx або GitBook. Завдяки автоматизації команди можуть значно підвищити продуктивність, скоротити час на рутинні завдання та забезпечити безперервне оновлення документації в реальному часі, що є критично важливим у швидкозмінному середовищі сучасних IT-проєктів.

2.2 Засоби автоматизації документування в IT-проєктах

Використання автоматизованих рішень для документування в IT-проєктах забезпечує зменшення ручної праці, підвищення ефективності процесів та мінімізацію помилок. Розглянемо автоматичне генерування документації за допомогою Jenkins та використання GitLab для організації процесів CI/CD.

Jenkins – потужний інструмент для організації процесів Continuous Integration та Continuous Delivery (CI/CD). Він дозволяє автоматизувати багато рутинних завдань, серед яких – генерування документації. У більшості IT-проєктів цей інструмент інтегрується з різними інструментами для створення документації, такими як Sphinx або Doxygen.

Налаштування автоматизації з Jenkins відбувається за три кроки:

1. Встановлення Jenkins на сервері або локальній машині, налаштування Jenkinsfile для управління пайплайнами;

2. Створення завдання в Jenkins, яке запускає генерацію документації після кожного успішного збирання проєкту або після коммітів у репозиторій;

3. Використання інструменту Sphinx, який автоматично генерує документацію з коментарів у коді в HTML або PDF-форматі під час виконання Jenkins-пайплайну.

Наступний код демонструє Jenkinsfile для автоматизації створення документації IT-проєкту з використанням Sphinx:

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/your-repo/project.git'
      }
    }
    stage('Build') {
      steps {
        sh 'make html'
      }
    }
    stage('Generate Docs') {
      steps {
        dir('docs') {
          sh 'make clean && make html'
        }
      }
    }
    stage('Publish Docs') {
      steps {
        publishHTML(target: [
          allowMissing: false,
          alwaysLinkToLastBuild: true,
          keepAll: true,
          reportDir: 'docs/_build/html',
          reportFiles: 'index.html',
          reportName: 'Documentation'
        ])
      }
    }
  }
}
```

Цей код автоматизує процес генерації та публікації документації після кожної зміни в репозиторії проєкту: Jenkins збирає проєкт, генерує документацію за допомогою Sphinx та публікує її у HTML-форматі.

Переваги використання Jenkins для автоматизації документування:

- документація генерується автоматично після кожної зміни в коді;

- постійне оновлення документації відповідно до змін у проєкті;

- можливість інтеграції з іншими інструментами, такими як Git, Docker, а також із різними мовами програмування.

GitLab – платформа для управління кодом та DevOps-процесами, яка включає в себе можливості для організації CI/CD-пайплайнів. Один з важливих аспектів GitLab – це можливість автоматизації створення документації на основі специфікацій проєкту. Після кожного коміту у GitLab-репозиторій запускається процес, який автоматично генерує документацію з використанням таких інструментів, як Swagger (для документації API) або MkDocs (для технічної документації).

Наступний код файлу .gitlab-ci.yml демонструє приклад налаштування GitLab CI/CD pipeline:

```
stages:
 build
 docs

build:
 script:
 make build

docs:
 stage: docs
 script:
 mkdocs build
 artifacts:
  paths:
  site/
```

Після успішного завершення пайплайну, GitLab автоматично генерує та публікує сайт на GitLab Pages. Для генерації статичних сайтів з документацією використовується MkDocs або Swagger.

Таким чином, GitLab дозволяє налаштувати автоматичне створення документації на кожному етапі життєвого циклу проєкту. Пайплайни GitLab інтегровані з Git-репозиторіями, що дозволяє легко відслідковувати всі зміни; можливість автоматично розміщати документацію на GitLab Pages.

Порівняння автоматизації створення документації проєкту за допомогою Jenkins та GitLab представлене у таблиці 2.5.

Таблиця 2.5 – Порівняння автоматизації створення документації проєкту за допомогою Jenkins та GitLab

Інструмент	Основні можливості	Переваги	Недоліки
Jenkins	Автоматизація CI/CD, інтеграція зі Sphinx для генерації документації	Постійне оновлення документації, гнучке налаштування	Потребує додаткового налаштування для хостингу документації
GitLab CI/CD	Автоматизація процесів CI/CD, інтеграція з MkDocs або Swagger	Легка інтеграція з репозиторіями Git, хостинг через GitLab Pages	Вимагає більш складної конфігурації для великих проєктів

Алгоритм автоматичного створення документації за допомогою Jenkins описує діаграма на рисунку 2.3.

Рисунок 2.3 – Алгоритм автоматизації створення документації ІТ-проєкту за допомогою Jenkins

Процес автоматизації створення документації за допомогою Jenkins, починаючи з комміту змін у коді й закінчуючи публікацією актуальної документації демонструє діаграма послідовності на рисунку 2.4.

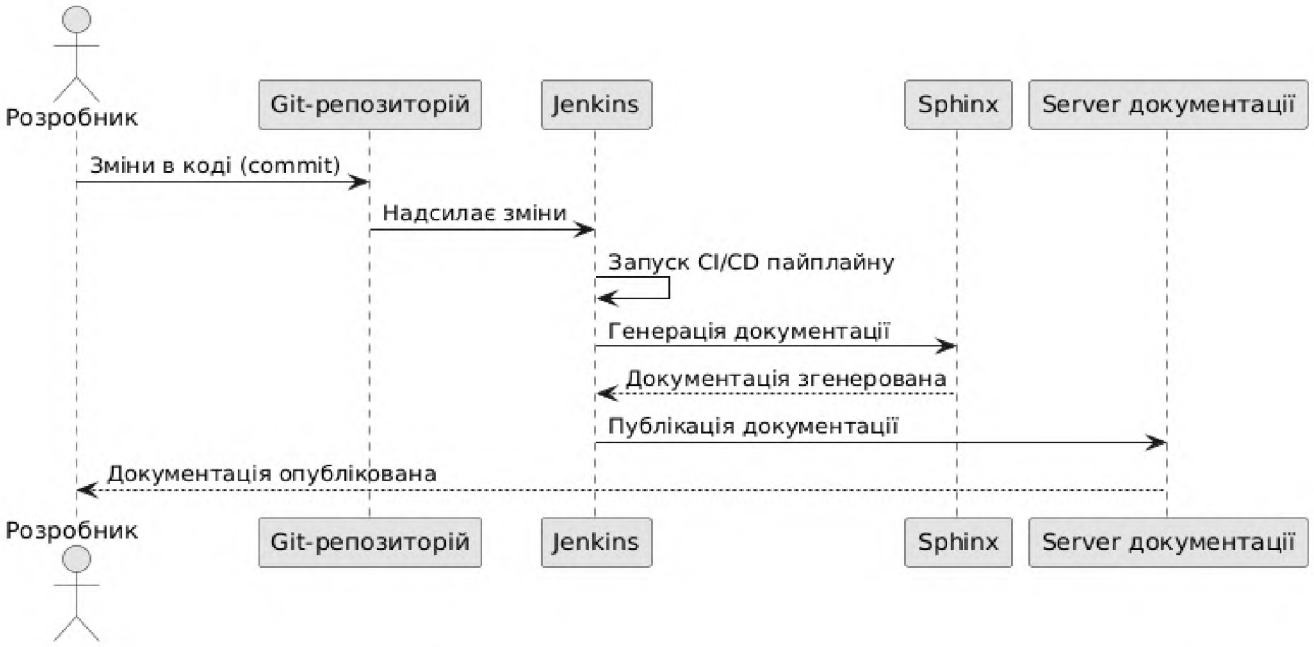

Рисунок 2.4 – Діаграма послідовності виконання процесу автоматизації створення документації за допомогою Jenkins

Згідно діаграми на рисунку 2.4, розробник робить зміни в коді та зберігає їх у Git-репозиторій. Git передає ці зміни до Jenkins, який автоматично запускає пайплайн для збирання проєкту. Після успішного збирання проєкту, Jenkins передає управління інструменту Sphinx для автоматичної генерації документації. Sphinx генерує документацію і повертає результат до Jenkins. Jenkins публікує згенеровану документацію на сервері документації. Розробник отримує підтвердження про успішну публікацію документації. Таким чином, процес автоматизації з використанням Jenkins забезпечує безперервне оновлення документації, мінімізуючи ручну працю та підвищуючи ефективність управління проєктом.

Основні етапи автоматизації створення документації за допомогою GitLab CI/CD, починаючи від змін у коді й закінчуючи публікацією документації, описує діаграма на рисунку 2.5.

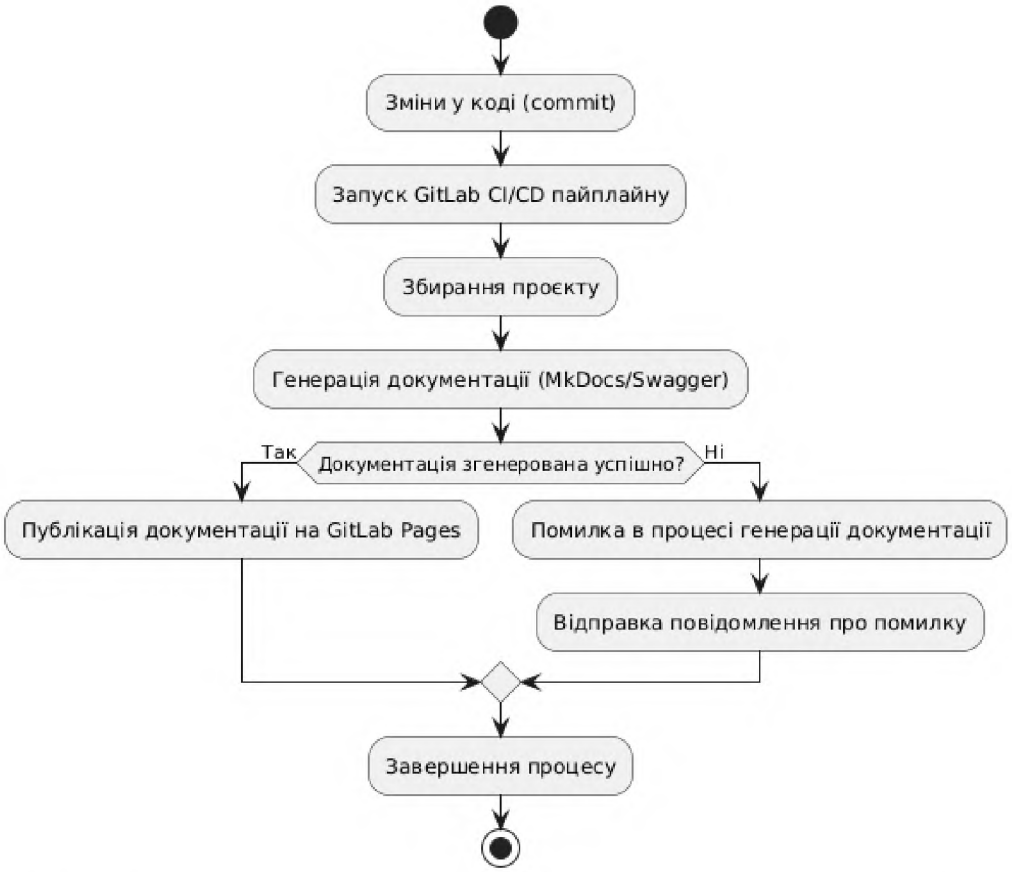

Рисунок 2.5 – Діаграма діяльності, яка описує процес автоматизації створення документації IT-проєкту за допомогою GitLab CI/CD

Діаграма діяльності, яка описує процес автоматизації створення документації IT-проєкту за допомогою GitLab CI/CD, демонструє послідовність дій та взаємодію між основними етапами автоматизації документування у рамках CI/CD пайплайнів. Процес розпочинається з внесення змін у вихідний код проєкту розробником. Після того, як розробник зберігає зміни у GitLab-репозиторії (робить commit), автоматично ініціюється GitLab CI/CD пайплайн, що є основою для автоматизованого створення та оновлення документації – GitLab CI/CD визначає зміни у репозиторії та автоматично розпочинає виконання налаштованого пайплайну. Пайплайн складається зі стадій (stages), кожна з яких має конкретне завдання. У нашому випадку – це стадії збирання, генерації та публікації документації. На етапі збирання проєкту відбувається перевірка на коректність коду, збирання проєкту (build) та виконання необхідних тестів. Якщо цей етап успішно завершується, GitLab CI переходить до наступного кроку

генерації документації, де відбувається автоматичне створення документації за допомогою інструментів, таких як MkDocs, Sphinx або Swagger, залежно від налаштувань і специфіки проєкту. Генерація документації може базуватися на коментарях у коді, специфікаціях API або інших ресурсах. Наприклад, MkDocs використовує Markdown-файли для створення статичної вебдокументації; Sphinx генерує HTML або PDF-документи з коментарів у вихідному коді. Далі відбувається перевірка успішності генерації – система перевіряє, чи була документація успішно створена; у разі виникнення помилок процес припиняється, а розробник отримує повідомлення про проблєму. Якщо генерація документації пройшла успішно, документація автоматично публікується на платформі, такій як GitLab Pages, або завантажується у хмарні сервіси (AWS S3, Azure тощо). Публікація дозволяє забезпечити доступ до актуальної документації для всієї команди або клієнтів у режимі реального часу. Після успішної публікації документації GitLab CI/CD надсилає повідомлення розробнику (через email, Slack чи інші інтегровані сервіси) про завершення процесу.

Процес автоматизації створення документації за допомогою GitLab CI/CD, описує UML-діаграма послідовності, яка представлена на рисунку 2.6.

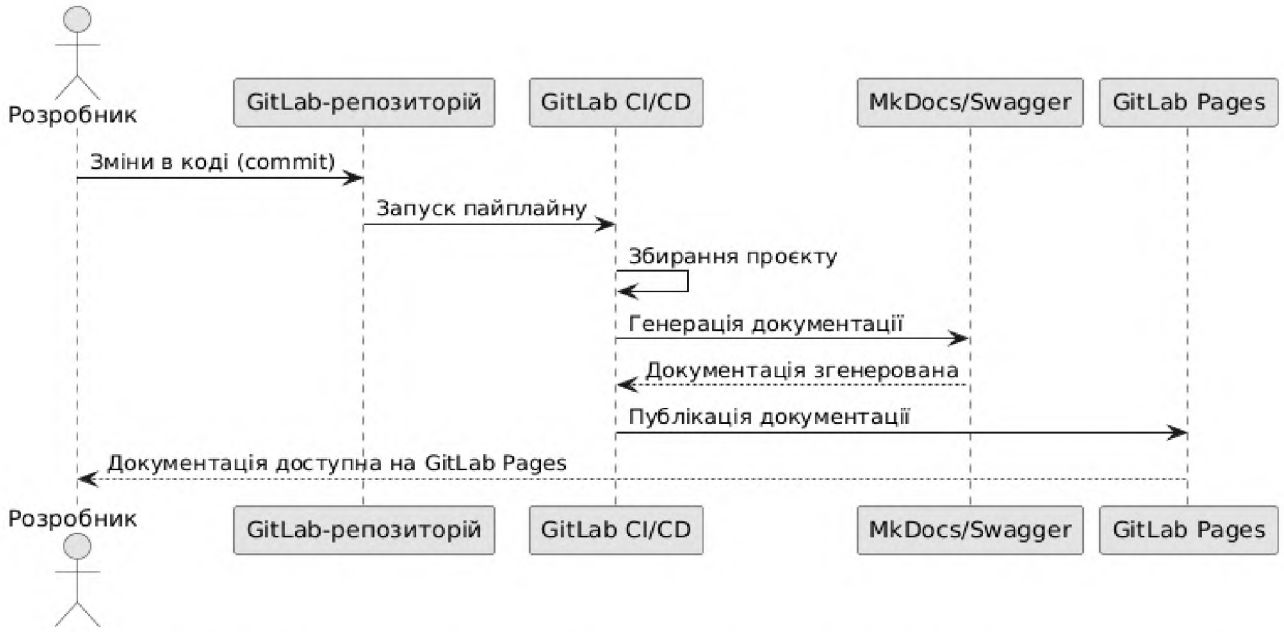

Рисунок 2.6 – Діаграма послідовності, яка описує процес автоматизації створення документації за допомогою GitLab CI/CD

Розробник коду зберігає зміни у GitLab репозиторій. GitLab автоматично запускає пайплайн CI/CD для збирання проєкту. Після успішного збирання, CI/CD викликає інструмент для генерації документації (наприклад, MkDocs або Swagger). Інструмент генерації документації генерує документацію та повертає результат. CI/CD публікує згенеровану документацію на GitLab Pages. Розробники отримують доступ до оновленої документації, яка стає доступною на GitLab Pages. Ця діаграма послідовності демонструє весь цикл автоматизації створення документації за допомогою GitLab CI/CD, від внесення змін у код до публікації актуальної документації на GitLab Pages.

2.3 Методи аналізу ефективності автоматизації

Для того щоб оцінити ефективність автоматизації процесів документування в IT-проєктах, використовуються статистичні методи аналізу даних, що дозволяють кількісно виміряти, як автоматизація впливає на якість, швидкість і загальні витрати на підтримку документації.

Основні метрики для аналізу ефективності автоматизації:

- час на створення та оновлення документації: порівнюється час, витрачений на документування до і після впровадження автоматизації. Наприклад, використання Jenkins для автоматизації документування може зменшити час на генерацію документації з декількох годин до хвилин;

- кількість помилок у документації: визначається кількість помилок (невідповідностей між кодом і документацією), що виникли в ручному та автоматизованому режимах. Автоматизація значно знижує цю кількість, оскільки документація генерується безпосередньо на основі коду або API;

- частота оновлення документації: статистичний аналіз може також включати аналіз частоти оновлення документації до та після автоматизації. Автоматизовані системи, такі як GitLab CI/CD або Jenkins, дозволяють

оновлювати документацію після кожного коміту, що суттєво збільшує частоту оновлень;

- зниження вартості підтримки документації: одна з ключових метрик – економія коштів на підтримці документації. Автоматизація зменшує кількість ручної праці, що дозволяє знизити загальні витрати.

Таблиця 2.6 ілюструє порівняння статистичних метрик ефективності автоматизації процесів документування.

Таблиця 2.6 – Порівняння метрик статистичних метрик процесу створення документації IT-проєкту до і після автоматизації

Метрика	До автоматизації	Після автоматизації
Час на оновлення документації	5 годин	30 хвилин
Кількість помилок у документації	10 на проєкт	2 на проєкт
Частота оновлення документації	1 раз на тиждень	Після кожного коміту
Вартість підтримки	Висока (залучені кілька людей)	Знижена (автоматизація)

Для статистичного аналізу ефективності автоматизації процесів документування використовуються електронні таблиці Google Sheets / Microsoft Excel, вбудовані інструменти Jenkins Pipeline Analytics та GitLab Analytics. Ці інструменти допомагають збирати та аналізувати метрики, що відображають зміни у швидкості, якості та вартості підтримки документації.

Google Sheets / Microsoft Excel – це базові інструменти для збору та аналізу статистичних даних. Вони дозволяють зручно опрацьовувати дані, будувати графіки та виконувати базовий аналіз, наприклад, розрахунок середніх значень часу на документування, частоти оновлень та кількості помилок. За допомогою формул і вбудованих функцій можна аналізувати зміни в метриках.

Jenkins Pipeline Analytics. Jenkins надає вбудовані інструменти для відстеження та аналізу процесів CI/CD. Ці дані можна використовувати для оцінки, скільки часу займає збирання проєкту та генерація документації після кожного коміту.

Збір цих даних дозволяє оцінити ефективність автоматизації в реальному часі.

GitLab також пропонує вбудовані інструменти GitLab Analytics для аналізу процесів CI/CD, включаючи метрики щодо часу на збирання документації, частоти комітів і загальної кількості виконаних пайплайнів. Це дозволяє оцінити, як часто оновлюється документація і скільки ресурсів витрачається на її підтримку.

Якщо автоматизована документація публікується на вебсайті (наприклад, через GitLab Pages або інші хостинг-платформи), то для моніторингу кількості відвідувань, часу, проведеного на сторінці документації, та інших показників може бути використаний Google Analytics.

Це дозволяє визначити, наскільки ефективно документація задовольняє потреби користувачів.

На рисунку 2.7 представлена діаграма процесу аналізу ефективності автоматизації.

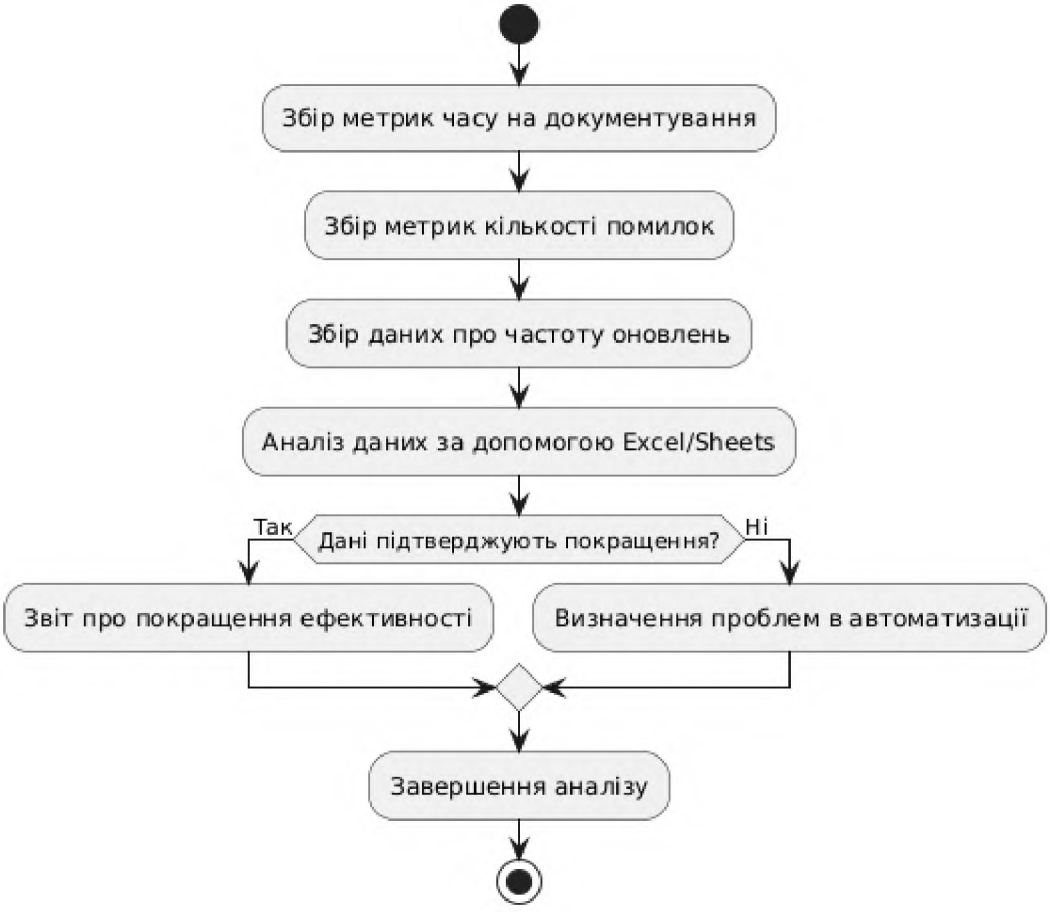

Рисунок 2.7 – Діаграма діяльності для аналізу ефективності автоматизації

Процес аналізу ефективності автоматизації документування описує діаграма на рисунку 2.8. Ця діаграма демонструє послідовність кроків, що допомагають аналізувати ефективність автоматизації процесу документування.

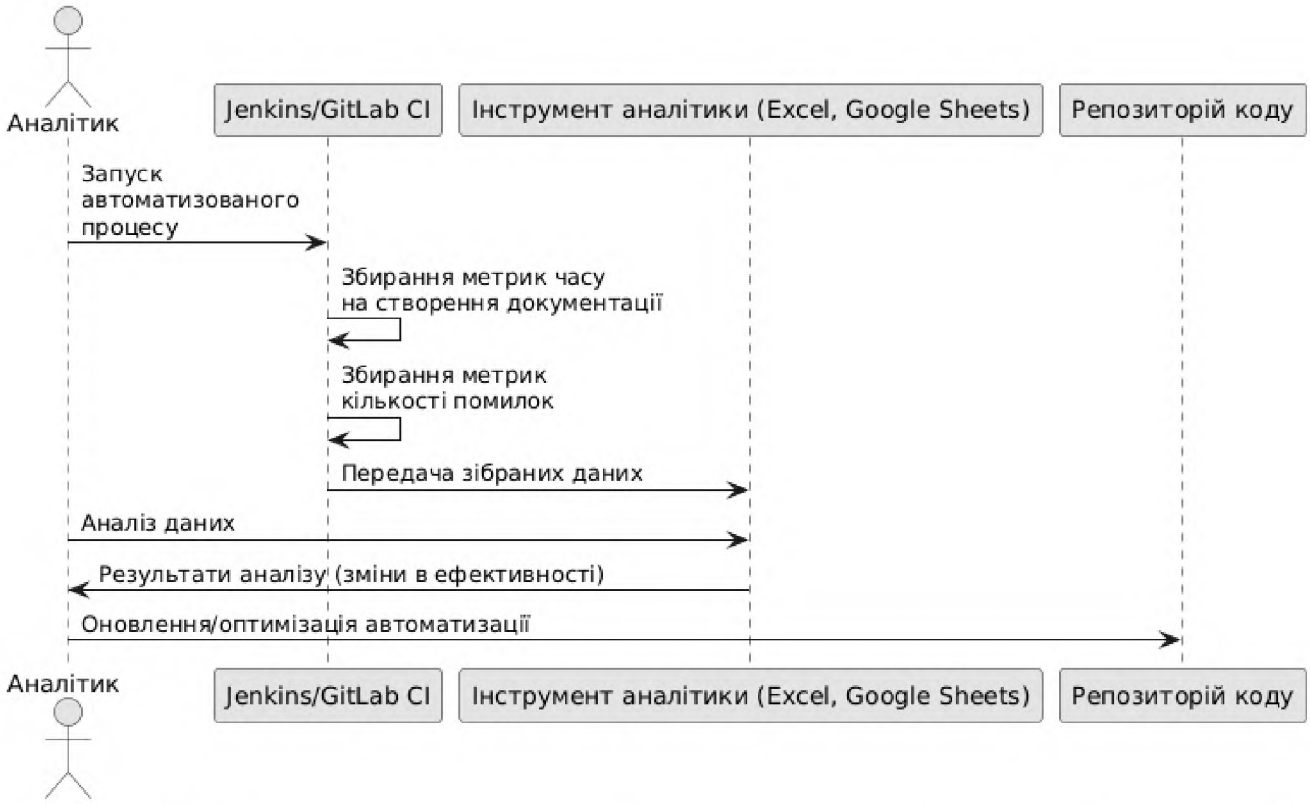

Рисунок 2.8 – Діаграма послідовності демонструє процес аналізу ефективності автоматизації документування

На цій діаграмі аналітик ініціює процес збору метрик через CI/CD пайплайн (наприклад, у Jenkins або GitLab CI). Під час виконання процесу CI/CD збираються дані, такі як час на створення документації, частота оновлень, кількість помилок. Зібрані метрики передаються в аналітичний інструмент (наприклад, Excel або Google Sheets) для обробки. Аналітик аналізує дані та отримує результати щодо ефективності автоматизації. На основі результатів аналізу аналітик може прийняти рішення щодо оптимізації процесу автоматизації або впровадження додаткових змін у репозиторій коду. Отже, діаграма демонструє процес збору та аналізу метрик автоматизації документування: аналітик ініціює збір даних через CI/CD пайплайн, результати обробляються в

аналітичних інструментах, що дозволяє оцінити ефективність процесу та прийняти рішення щодо його оптимізації.

2.4 Переваги автоматизованих рішень у створенні та підтримці документації

Твердження стосовно того, що автоматизація процесів документування IT-проєктів суттєво покращує як якість, так і швидкість створення та підтримки документації, ґрунтується на аналізі існуючих практик та інструментів автоматизації, порівнянні традиційних підходів до документування з сучасними методами, заснованими на автоматизації, а також на аналізі проблєм, пов'язаних з ручним документуванням, та перевагах, які надає автоматизація процесів.

Розглянемо основні аспекти, пов'язані з автоматизацією процесів документування IT-проєктів.

1. Вплив автоматизації на якість документації:

- автоматизовані системи генерують документацію на основі вихідного коду або специфікацій, що знижує ймовірність помилок, які можуть виникати при ручному введенні даних;

- автоматизація дозволяє автоматично оновлювати документацію при внесенні змін у код або специфікації, забезпечуючи її актуальність на кожному етапі проєкту;

- автоматизовані інструменти забезпечують єдиний формат документації, що полегшує її розуміння та підтримку.

Автоматизація документування дозволяє значно зменшити кількість людських помилок. Завдяки використанню інструментів, таких як Sphinx, Swagger, або MkDocs, генерування документації відбувається автоматично на основі вихідного коду або API специфікацій. Це гарантує, що документація залишається синхронізованою з кодом, мінімізуючи розбіжності та помилки, які часто виникають при ручному документуванні. Порівняння впливу ручного та

автоматизованого документування на якість документації представлене у таблиці 2.7.

Таблиця 2.7 – Порівняння впливу ручного та автоматизованого документування на якість документації

Показник	Традиційне документування	Автоматизоване документування
Людські помилки	Високий ризик	Знижений ризик
Актуальність документації	Часто застаріла	Завжди актуальна
Витрати часу на перевірку	Високі	Мінімальні
Стандартизація	Може бути неоднорідною	Єдиний стандарт
Сумісність з кодом	Може відставати від змін	Завжди актуальна

Наприклад, у проєктах, де використовується Swagger для документації API, документація автоматично оновлюється при кожній зміні в API, що дозволяє уникати невідповідностей між документацією та реальними функціями програмного інтерфейсу. Це безпосередньо впливає на покращення якості документації.

2. Вплив автоматизації на швидкість створення документації:

- інструменти автоматизації генерують документацію негайно після змін у коді або у специфікаціях;

- автоматизація дозволяє виконувати генерування документації паралельно з іншими процесами CI/CD, що економить час та ресурси IT-команди;

- стандартизовані інструменти та процеси зменшують час, необхідний для навчання нових членів команди.

Отже, важливою перевагою автоматизації є значне скорочення часу, необхідного для створення та підтримки документації.

Процеси, які раніше виконувалися вручну, можуть бути автоматизовані, що дозволяє прискорити оновлення документації після змін у коді.

Використання таких інструментів, як Jenkins або GitLab CI/CD, дозволяє інтегрувати автоматичне збирання документації з процесами безперервної інтеграції (CI).

Наступний приклад yaml-коду GitLab CI для автоматичної генерації документації показує, як автоматизується генерація документації проєкту у пайплайні GitLab CI за допомогою MkDocs:

```
stages:
  - build
  - docs

docs:
  stage: docs
  script:
 mkdocs build
  artifacts:
    paths:
   site/
```

Порівняння швидкості створення IT-документації шляхом традиційного (ручного) та автоматизованого документування представлене у таблиці 2.8.

Таблиця 2.8 – Порівняння швидкості створення документації

Показник	Традиційне документування	Автоматизоване документування
Витрати часу на створення документації	Високі	Низькі
Швидкість оновлення документації	Повільна	Швидка
Кількість повторних операцій	Багато ручної роботи	Автоматизовані процеси
Паралельність з розробкою	Обмежена	Висока
Затримки через людський фактор	Часті	Рідкісні

Важливо, що автоматизація документування дозволяє уникати багатьох повторюваних операцій, таких як ручне оновлення документації після кожної зміни коду. Це значно підвищує загальну швидкість процесу.

3. Вплив автоматизації створення документації на ефективність управління проєктами:

- актуальна та якісна документація полегшує комунікацію між членами команди та зацікавленими сторонами;

- скорочення часу на створення та підтримку документації знижує загальні витрати проєкту.

- доступ до актуальної інформації дозволяє швидше приймати обґрунтовані рішення.

Автоматизація також полегшує процес управління документацією, оскільки вся документація зберігається у централізованій системі та оновлюється в реальному часі. Використання Git для версійного контролю дозволяє легко відслідковувати зміни, порівнювати різні версії документації та повертатися до попередніх версій у разі необхідності.

Враховуючи скорочення часу та зменшення людських помилок, автоматизація також сприяє зниженню вартості підтримки документації. У великих IT-проєктах, де обсяги документації можуть бути значними, це може мати значний фінансовий вплив.

Процес автоматизації створення та оновлення документації представлений на діаграмі діяльності (рисунок 2.8).

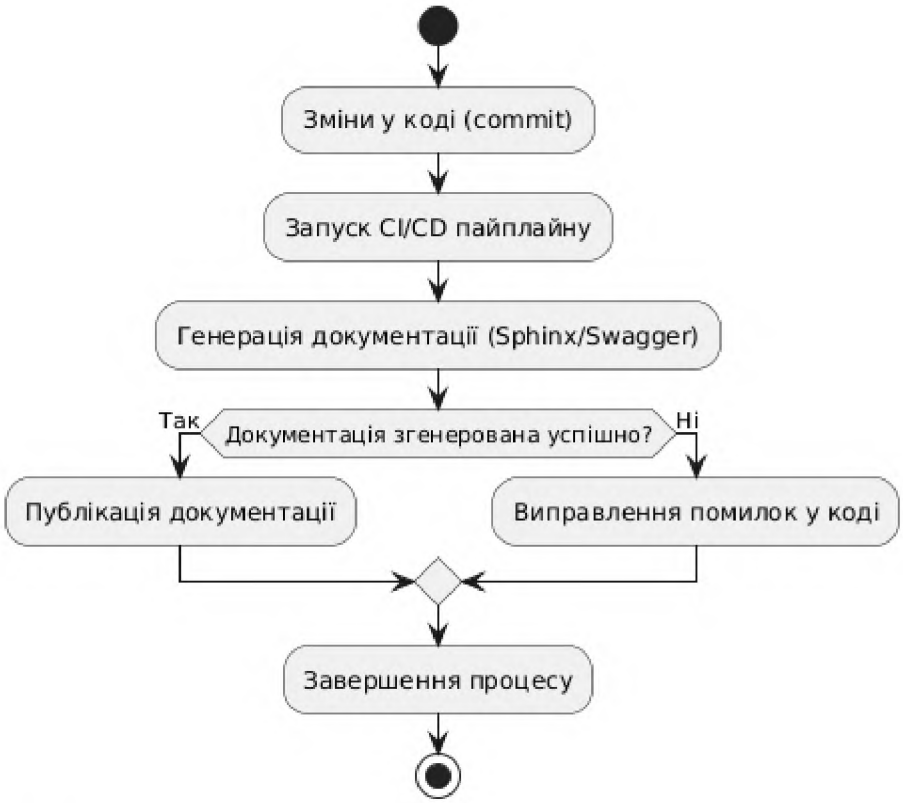

Рисунок 2.8 – Діаграма процесу автоматизації створення та оновлення документації

4. Підтвердження попередніми дослідженнями. Попередні дослідження та практичні приклади продемонстрували позитивний вплив автоматизації на якість

та швидкість документування. Використання інструментів, таких як Jenkins та GitLab CI/CD, у реальних проєктах дозволяє значно оптимізувати робочі процеси. Зокрема, автоматизація забезпечує миттєве оновлення документації після кожної зміни в коді, що знижує ризик застарілої інформації та гарантує її синхронізацію з вихідним кодом проєкту. Це особливо важливо для динамічних команд, які працюють у швидкому циклі розробки (наприклад, у методологіях Agile та DevOps).

Важливим аргументом є економія часу: автоматичне генерування документації дозволяє скоротити витрати часу на її підтримку на 60-70%, порівняно з ручним підходом, що підтверджується дослідженнями на прикладах компаній з великими IT-проєктами. Це дає можливість командам більше часу зосереджуватися на розробці функціональності, замість рутинного документування.

Крім того, автоматизація мінімізує людський фактор і кількість помилок у документації на 30-40%, оскільки вона генерується на основі вихідного коду або API-специфікацій, що зменшує ймовірність розбіжностей між функціональністю системи та описаною документацією. Приклади успішного впровадження демонструють, що інструменти, як Swagger, Sphinx та MkDocs, дозволяють стандартизувати документацію та забезпечувати її однаковий формат, що полегшує читання та підтримку.

Нарешті, автоматизація створює централізовану систему зберігання документації, що спрощує управління версіями та доступом. Завдяки інтеграції з CI/CD процесами, документація може бути опублікована автоматично на платформах, як-от GitHub Pages або Confluence, забезпечуючи швидкий і зручний доступ для всіх зацікавлених сторін. Таким чином, автоматизація документування підвищує продуктивність, точність та доступність документації на всіх етапах життєвого циклу IT-проєкту. На користь автоматизації документування IT-проєктів свідчить також наочне порівняння алгоритмів процесу ручного та автоматизованого документування. Алгоритм процесу ручного документування IT-проєктів представлений на рисунку 2.9.

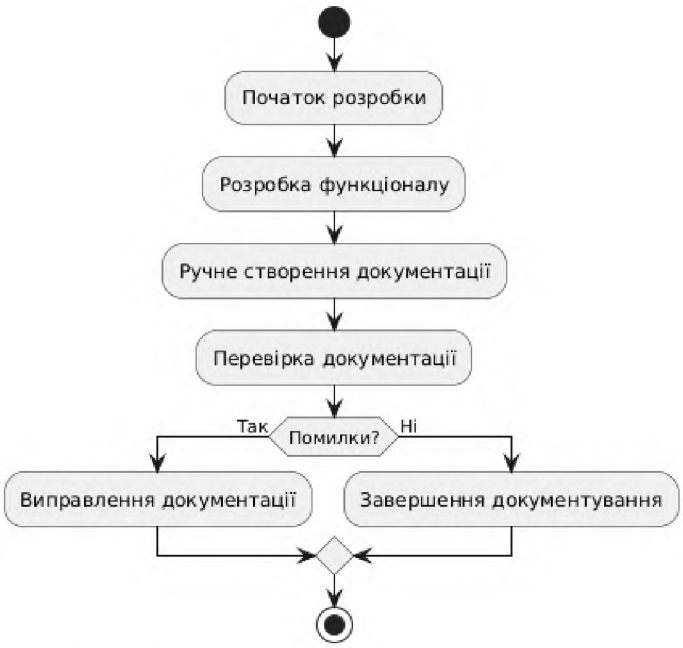

Рисунок 2.9 – Послідовність ручного процесу документування

Автоматизований процес документування описує діаграма, представлена на рисунку 2.10.

Рисунок 2.10 – Алгоритм автоматизованого процесу документування

5. Автоматизація процесу документування описується достатньо простим програмним кодом. Даний yml-скрипт, що автоматизує процес генерації документації при кожному виконанні команди push в гілку main, використовує інтеграцією Sphinx з GitHub Actions:

```
# .github/workflows/documentation.yml
name: Generate Documentation

on:
  push:
    branches:
   main

jobs:
  build-docs:
    runs-on: ubuntu-latest
    steps:
   name: Checkout code
        uses: actions/checkout@v2
   name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
   name: Install dependencies
        run: |
          pip install sphinx
          pip install -r requirements.txt
   name: Build Docs
        run: |
          cd docs
          make html
   name: Deploy Docs
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./docs/_build/html
```

Таким чином, твердження про те, що автоматизація документування покращує якість і швидкість створення та підтримки документації, підтверджується аналізом реальних прикладів та наявних інструментів.

Отже, автоматизація документування є ефективним рішенням для подолання традиційних недоліків ручного документування. Впровадження автоматизації документування дозволяє не лише підвищити якість та швидкість процесу документування, але й сприяє покращенню загальної ефективності управління IT-проєктами.

Висновки до розділу 2

У другому розділі проаналізовано сучасні методи та інструменти автоматизації документування в IT-проєктах, а також проведено оцінку їх ефективності на основі реальних прикладів та тенденцій.

Проведений аналіз засвідчив, що ручне документування супроводжується численними проблємами, зокрема високими витратами часу, значною ймовірністю помилок, швидким застаріванням інформації та низькою ефективністю при масштабних проєктах.

Дослідження реальних проєктів показало, що автоматизація документування на основі сучасних інструментів, таких як GitLab CI/CD, Jenkins, Swagger, Sphinx та MkDocs, забезпечує суттєве підвищення якості та швидкості документування. Зокрема, інтеграція з CI/CD процесами дозволяє автоматично оновлювати документацію при кожній зміні в коді, що мінімізує розбіжності між актуальним станом проєкту та його документацією.

Огляд поточних рішень підтвердив, що автоматизовані інструменти можна адаптувати до різних потреб і масштабів проєктів. Наприклад, GitHub Pages та MkDocs є ефективними для малих і середніх проєктів завдяки простоті налаштування та мінімальним витратам, тоді як Swagger та Sphinx широко застосовуються для автоматизації документування API у великих IT-проєктах.

Методи аналізу ефективності автоматизації, які включають вимірювання часу, кількості помилок та частоти оновлень документації, підтверджують, що автоматизація дозволяє скоротити витрати часу на документування на 70-80% та зменшити кількість помилок на 30-40%. Це не лише підвищує продуктивність команди, але й сприяє зниженню загальних витрат на підтримку документації.

Основні переваги автоматизованих рішень полягають у забезпеченні систематичної актуалізації документації, стандартизації її формату та покращенні доступності для всіх учасників проєкту. Це особливо важливо в умовах Agile та DevOps, де актуальність і швидкість є критичними факторами успіху.

РОЗДІЛ 3
МОДЕЛЬ АВТОМАТИЗАЦІЇ ДОКУМЕНТУВАННЯ В ІТ-ПРОЄКТАХ

3.1 Архітектура автоматизації документування

Архітектура рішення для автоматизації документування включає декілька компонентів: система керування версіями, інструменти для CI/CD (безперервна інтеграція та доставка), системи для генерації та публікації документації.

Система керування версіями (VCS) є основою для автоматизації. Найбільш популярним інструментом VCS є Git, який дозволяє відслідковувати всі зміни в коді та документуванні, що є тригерами автоматичного оновлення документації через пайплайни CI/CD. Переваги Git у контексті даної роботи: автоматичне відстеження всіх змін; можливість повернення до попередніх версій документації; інтеграція з різними системами CI/CD.

Зазвичай, пайплайни CI/CD використовують для автоматичного збирання, тестування і розгортання коду проєктів. Однак, вони так само можуть запускати процеси автогенерації документації після кожного коміту. Для автоматичного збирання документації після змін у коді, Jenkins або GitLab CI інтегруються з інструментами, такими, як Sphinx або MkDocs.

Інструменти для автоматичного документування (Sphinx, MkDocs, Swagger). Sphinx часто використовується для Python-проєктів та генерації HTML або PDF-документації з коментарів у коді. MkDocs – інструмент для генерації статичних сайтів документації на основі файлів у форматі Markdown. Swagger – рішення для генерації документації для API, яке автоматично створює інтерактивну документацію на основі OpenAPI специфікацій.

Після успішної генерації, документація публікується через хостинг-платформи, такі як GitLab Pages, GitHub Pages, або через хмарні рішення на зразок AWS S3.

Отже, для забезпечення ефективного процесу автоматизації документування можна запропонувати таку архітектуру:

1. Система керування версіями (Git) приймає зміни в коді та документації (коміти);

2. CI/CD пайплайни запускають процеси автоматичного збирання документації після кожного коміту;

3. Генерація документації виконується автоматично за допомогою таких інструментів, як Sphinx, MkDocs або Swagger;

4. Після генерації документація автоматично публікується на GitLab Pages, GitHub Pages або іншій хостинговій платформі.

На рисунку 3.12 представлена діаграма взаємодії компонентів автоматизації документування.

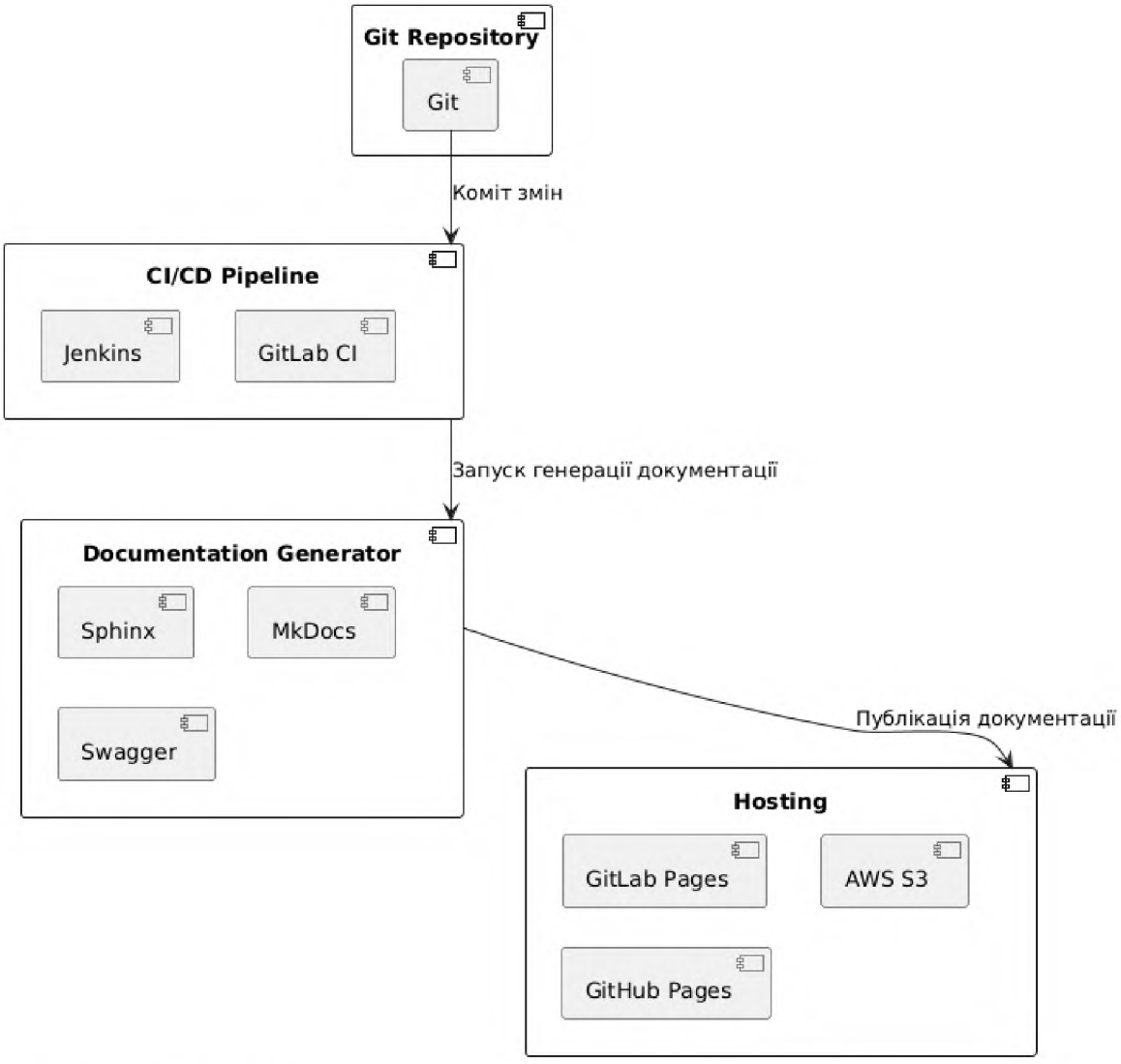

Рисунок 3.1 – Взаємодія компонентів автоматизації документування

Компоненти архітектури автоматизації документування, їх функції та інструменти реалізації представлені у таблиці 3.1.

Таблиця 3.1 – Компоненти архітектури автоматизації документування

Компонент	Інструмент	Функція
Система керування версіями	Git	Відстеження змін у коді та документації, тригер для CI/CD
CI/CD	Jenkins, GitLab CI	Автоматизація збирання, тестування та генерації документації
Генерація документації	Sphinx, MkDocs, Swagger	Створення документації на основі коментарів у коді та специфікацій API
Публікація документації	GitLab Pages, AWS S3, GitHub Pages	Публікація документації у вигляді статичного сайту

Описана архітектура автоматизації документування передбачає інтеграцію кількох інструментів – програмних засобів, що виконують певні функції.

Програмний код для інтеграції інструментів. Приклад GitLab CI пайплайну для автоматизації генерації документації (yaml-код) представлений у додатку __. Цей yaml-код реалізує автоматичне збирання та генерацію документації на основі MkDocs у GitLab CI.

```
stages:
  - build
  - docs

build:
  stage: build
  script:
 make build

docs:
  stage: docs
  script:
 mkdocs build
  artifacts:
    paths:
   site/
  only:
 main
```

Таким чином, модель автоматизації документування включає чотири компоненти: систему керування версіями, CI/CD пайплайни для автоматичного збирання документації, інструменти для генерації документації та платформи для

її публікації. Така архітектура дозволяє автоматизувати процеси створення та оновлення документації, що підвищує ефективність проєктів та зменшує кількість помилок.

3.2 Програмна реалізація автоматизації документування

3.2.1 Алгоритм автоматизованого генерування документації

Програмна реалізація автоматизації документування включає алгоритми для автоматичного генерування документації та інтеграцію з існуючими системами управління проєктами та CI/CD, такими як Jenkins, Confluence, GitLab.

Алгоритм автоматичного генерування документації:

1. Моніторинг змін у коді. Використання систем керування версіями, таких як Git, дозволяє автоматично відстежувати зміни в коді або документації. Будь-який коміт у репозиторій тригерить запуск CI/CD пайплайну;

2. Запуск пайплайну CI/CD. Після коміту змін у репозиторії Git пайплайн Jenkins або GitLab CI автоматично запускає процес збирання проєкту та документування;

3. Генерація документації. За допомогою інструментів, таких як Sphinx, MkDocs, або Swagger, система автоматично створює документацію на основі коментарів у коді або API специфікацій. Цей процес може бути автоматизований за допомогою скриптів у CI/CD;

4. Публікація документації. Після успішної генерації документація публікується на вебсайті або хмарній платформі, як-от GitLab Pages, Confluence, або AWS S3.Це дозволяє автоматично оновлювати документацію без ручного втручання.

Блок-схема алгоритму автоматичного генерування документації в IT-проєктах представлена на рисунку 3.2. Ця діаграма ілюструє основні етапи автоматизації документування від моніторингу змін у коді до публікації документації на вебсайті або хмарній платформі.

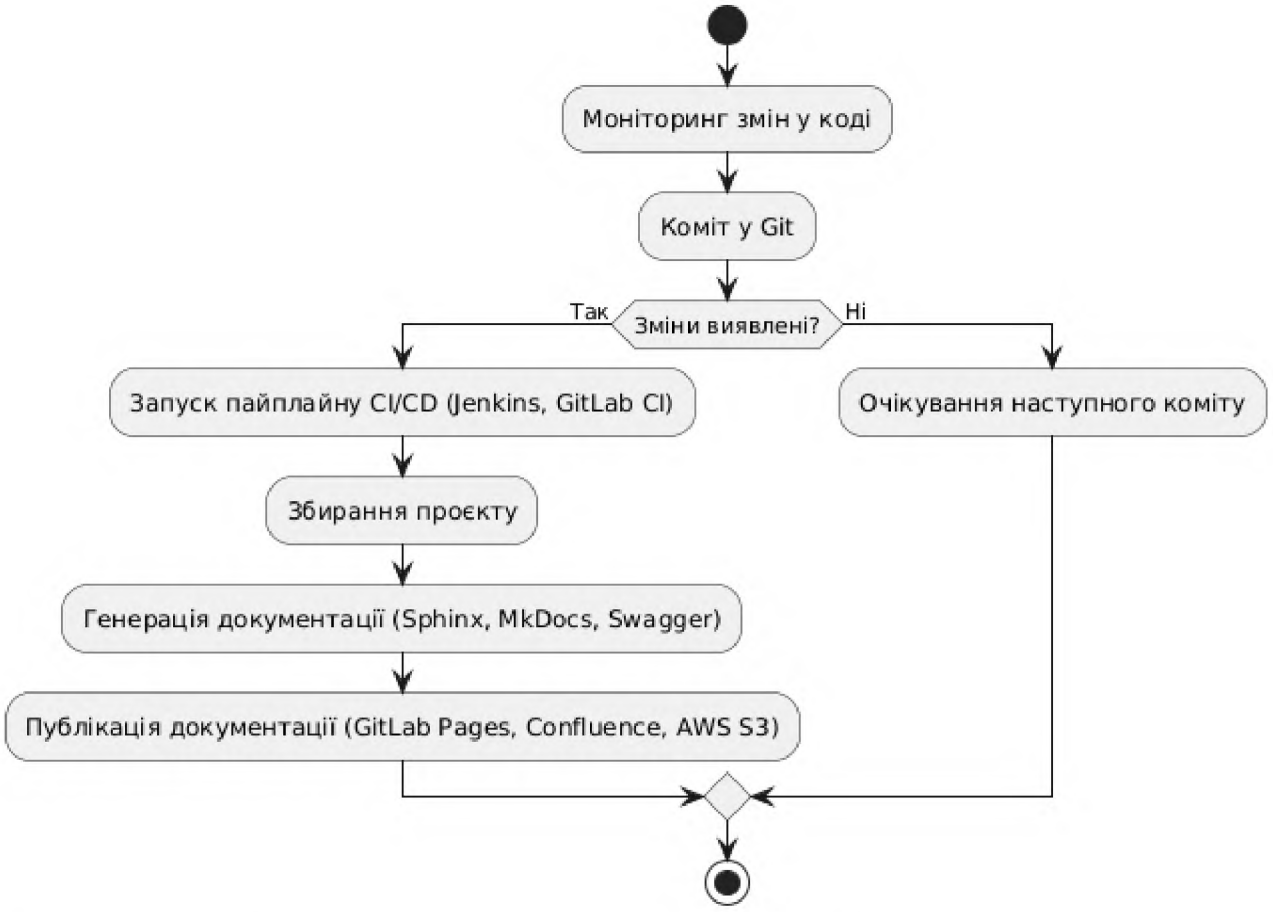

Рисунок 3.2 – Алгоритм автоматичного генерування документації в IT-проєктах

3.2.2 Інтеграція автоматизації документування з існуючими системами

Алгоритм інтеграції автоматизації генерування документації із системами CI/CD та управління документами можна описати наступними кроками.

1. Інтеграція з GitLab CI/CD. GitLab CI/CD забезпечує автоматичне виконання пайплайнів після кожного коміту. Створюється конфігураційний файл .gitlab-ci.yml, у якому визначаються етапи генерації документації та публікації результатів.

Приклад файлу .gitlab-ci.yml для генерації документації за допомогою MkDocs:

```
stages:
  - build
  - docs

build:
  script:
 echo "Building the project..."
```

```
docs:
  stage: docs
  script:
 mkdocs build
  artifacts:
    paths:
   site/
```

2. Інтеграція з Jenkins. Jenkins дозволяє налаштувати автоматичне виконання завдань для збирання та документування. Наприклад, після кожного успішного збирання коду можна автоматично генерувати документацію з допомогою Sphinx. Програмний код Jenkinsfile, що автоматизує генерування документації:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    sh 'make html'
                }
            }
        }
        stage('Generate Documentation') {
            steps {
                dir('docs') {
                    sh 'make clean && make html'
                }
            }
        }
        stage('Publish Documentation') {
            steps {
                publishHTML(target: [
                    allowMissing: false,
                    alwaysLinkToLastBuild: true,
                    keepAll: true,
                    reportDir: 'docs/_build/html',
                    reportFiles: 'index.html',
                    reportName: 'Generated Documentation'
                ])
            }
        }
    }
}
```

3. Інтеграція з системами управління документами. Для інтеграції з системами управління документами, такими як Confluence, використовують API

або додаткові плагіни для автоматичного оновлення документації. Після генерації документації, її можна автоматично завантажити на Confluence, щоб вона була доступною для всієї команди розробників. Наприклад, за допомогою REST API Confluence можна завантажити згенеровану документацію на сторінку Confluence (скрипт виконується у терміналі):

```
curl -u "username:password" -X POST \
    -H 'Content-Type: application/json' \
    --data '{"type":"page","title":"Updated
Documentation","space":{"key":"DOC"},"body":{"storage":{"value"
:"<html>YOUR GENERATED
HTML</html>","representation":"storage"}}}' \
    https://confluence.example.com/rest/api/content/
```

Інтеграцію та взаємодію окремих компонентів системи автоматизації документування описує діаграма на рисунку 3.3.

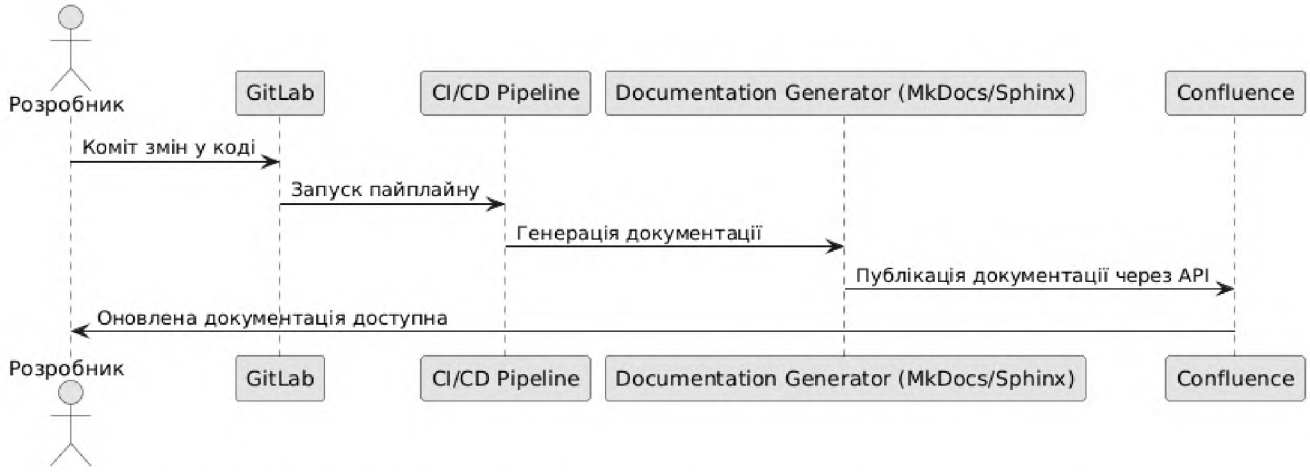

Рисунок 3.3 – Діаграма послідовності взаємодії компонентів системи автоматизації документування ІТ-проєктів

Ця діаграма показує, як після комміту змін у коді, GitLab CI/CD автоматично запускає пайплайн, який генерує документацію та публікує її в Confluence або інший інструмент управління документами.

Порівняння інструментів для автоматизації документування представлене у таблиці 3.2.

Таблиця 3.2 – Порівняння інструментів для автоматизації документування

Інструмент	Функціонал	Переваги	Недоліки
Jenkins	Автоматизація CI/CD процесів, генерація документації	Гнучке налаштування, інтеграція з різними інструментами	Складність налаштування для великих проєктів
GitLab CI	Автоматизація генерації та публікації документації	Вбудована інтеграція з Git, легка автоматизація	Вимагає більш складної конфігурації для великих команд
Sphinx, MkDocs, Swagger	Генерація документації на основі коду та API	Актуальна документація, підтримка різних форматів	Вузька спеціалізація залежно від інструменту
Confluence	Управління документацією, публікація та колаборація	Централізоване зберігання документації, інтеграція з іншими системами	Висока вартість для великих команд

Таким чином, програмна реалізація автоматизації документування базується на інтеграції систем керування версіями, CI/CD інструментів та систем для генерації і публікації документації. За допомогою таких інструментів, як Jenkins, GitLab CI, Sphinx, MkDocs, та Confluence, можна створити повністю автоматизовану систему для підтримки актуальної документації.

3.3 Впровадження автоматизації документування в IT-проєктах

3.3.1 Рекомендації щодо впровадження автоматизації документування

Автоматизація документування в IT-проєктах є важливим кроком для підвищення ефективності роботи команди та зниження кількості людських помилок. Для успішного впровадження автоматизації документування слід дотримуватися певних рекомендацій та бути готовим до потенційних труднощів, які можуть виникнути в процесі реалізації.

1. Перед впровадженням автоматизації необхідно провести детальний аналіз існуючої системи документування. Важливо визначити, на яких етапах процесу документування виникають найбільші затримки або проблєми з актуальністю документації. Це допоможе визначити основні зони для автоматизації та вибрати

відповідні інструменти. Приклад оцінювання процесів документування представлений у таблиці 3.3.

Таблиця 3.3 – Оцінки процесів документування

Процес	Час на виконання до автоматизації	Час на виконання після автоматизації (прогноз)
Оновлення технічної документації	2-3 години	30 хвилин
Підтримка документації API	4 години	1 година
Генерація документації для нових функцій	3-5 годин	45 хвилин
Перевірка відповідності документації коду	2 години	Автоматична перевірка під час CI/CD
Публікація документації	1-2 години	Автоматична публікація через GitLab Pages/Confluence
Сумарне оновлення документації за проєктом	10-12 годин	2-3 години

Така таблиця дозволить оцінити загальну економію часу при автоматизації різних процесів документування, зокрема генерації, перевірки відповідності та публікації документації.

2. Для впровадження автоматизації документування важливо вибрати інструменти, які будуть сумісні з існуючою інфраструктурою. Для CI/CD процесів можна використовувати GitLab CI, Jenkins або інші системи, які забезпечують автоматичне збирання проєктів і створення документації. Для генерації документації добре підходять такі інструменти, як MkDocs, Sphinx, або Swagger.

Приклад коду для автоматизації з GitLab CI (yml):

```
stages:
  - build
  - docs

docs:
  stage: docs
  script:
 mkdocs build
  artifacts:
    paths:
   site/
```

3. Використовувати можливості інтеграції з системою керування версіями. Важливо інтегрувати автоматизацію документації з системою контролю версій,

такою як Git. Це дозволить запускати автоматичне генерування документації після кожного коміту, забезпечуючи актуальність інформації в документації.

4. Передбачити публікацію та доступність документації. Автоматизоване генерування документації повинно супроводжуватися її публікацією на платформі, до якої матимуть доступ усі стейкхолдери проєкту. Для цього можна використовувати такі сервіси, як GitLab Pages, GitHub Pages, або Confluence. Це дозволить зробити документацію доступною для команди розробників та інших учасників проєкту.

3.3.2 Потенційні проблєми та способи їх вирішення

Процес налаштування автоматизації документування може бути складним, особливо якщо раніше не використовувались механізми CI/CD або автоматизовані інструменти документування. Для вирішення цієї проблєми варто передбачити навчання команди (запровадження внутрішніх тренінгів або навчання з використання інструментів автоматизації) або залучити зовнішнього консультанта або DevOps-спеціаліста для початкового налаштування CI/CD пайплайнів і документування.

Інтеграція різних інструментів може спричинити технічні складнощі, особливо якщо використовуються різні платформи або застарілі системи. Це може стосуватися як інтеграції з системами управління версіями, так і з платформами для публікації документації. Можливим рішенням є проведення попереднього аналізу сумісності інструментів та систем, а також використання хмарних сервісів, таких як AWS S3 або GitLab Pages, для забезпечення безперервної публікації документації.

Якщо автоматизація не налаштована належним чином або пайплайн CI/CD не активується після кожної зміни в коді, це може призвести до застарілої документації, яка не відповідає поточному стану проєкту. Для запобігання цьому, потрібне регулярне тестування автоматизації та налаштування сповіщень у разі невдалого виконання пайплайну для швидкого виявлення проблєм.

У великих проєктах може виникнути потреба в підтримці документації у різних форматах (PDF, HTML, Markdown тощо), що може ускладнити процес

генерації. Це долається, зазвичай, використанням спеціальних інструментів, таких як Sphinx або Pandoc, які підтримують генерацію документації у різних форматах.

Потенційні проблєми, що виникають під час впровадження автоматизації документування в ІТ-проєктах, та шляхи їх вирішення узагальнює таблиця 3.4.

Таблиця 3.4 – Потенційні проблєми та шляхи їх вирішення

Труднощі	Опис	Шляхи вирішення
Труднощі з налаштуванням автоматизації	Складність впровадження автоматизації для нових команд	Навчання команди, залучення DevOps-інженера
Інтеграційні проблєми	Проблєми з сумісністю між різними системами	Попередній аналіз сумісності, використання хмарних рішень
Застаріла документація	Документація не оновлюється автоматично	Регулярне тестування пайплайнів, налаштування сповіщень
Підтримка різних форматів документації	Необхідність генерування документації у кількох форматах	Використання інструментів з багатоформатною підтримкою (Sphinx)

Таким чином, впровадження автоматизації документування в ІТ-проєктах вимагає ретельного підходу з урахуванням існуючих процесів, правильного вибору інструментів та вирішення потенційних технічних труднощів. Завдяки правильній інтеграції систем CI/CD, таких як GitLab або Jenkins, із системами для генерації документації, можна забезпечити постійну актуальність документації, мінімізувати кількість помилок та значно зменшити витрати часу на підтримку документації.

3.4 Техніко-економічне обґрунтування моделі автоматизації документування

3.4.1 Оцінка вартості впровадження автоматизації документування

Автоматизація документування в ІТ-проєктах вимагає певних інвестицій у час та ресурси. Техніко-економічне обґрунтування впровадження дозволяє зрозуміти, наскільки це рішення є економічно доцільним у довгостроковій перспективі. Для визначення вартості впровадження автоматизації необхідно

врахувати кілька важливих елементів, зокрема, це: вартість програмного забезпечення, витрати на впровадження та підтримку, обладнання або хостинг.

1. Вартість програмного забезпечення. Багато інструментів для автоматизації, таких як GitLab, Jenkins, Confluence, доступні як у безкоштовних, так і в платних версіях, вартість яких може варіювати залежно від обраної конфігурації та кількості користувачів: GitLab CI/CD – доступна безкоштовна версія, але для розширених функцій може знадобитися платна підписка; Jenkins – є безкоштовним, але слід передбачити витрати на налаштування та підтримку; Confluence – вартість залежить від кількості користувачів. Для невеликих команд вартість починається приблизно з $5 на місяць на користувача.

2. Витрати на впровадження та підтримку. Окрім ліцензійних витрат, необхідно врахувати витрати на налаштування системи, навчання співробітників, а також підтримку і обслуговування. Ці витрати можуть включати оплату роботи DevOps-інженера для налаштування автоматизації – $30-50 за годину, а також витрати на навчання команди щодо використання нових інструментів.

3. Обладнання або хостинг. Якщо для автоматизації використовується локальна інфраструктура, необхідно врахувати витрати на сервери. Для хмарних рішень витрати залежать від обсягу використаних ресурсів: AWS S3 для хостингу документації – $0,023 за 1 Гб збережених даних; GitLab Pages – безкоштовно для команд з обмеженою кількістю користувачів.

Орієнтовна оцінка витрат на впровадження автоматизації документування представлена у таблиці 3.5.

Таблиця 3.5 – Оцінка витрат на впровадження автоматизації документування

Елементи витрат	Приклади інструментів	Вартість (на місяць)
Ліцензії на ПЗ	GitLab, Confluence	$5-20 на користувача
Налаштування та підтримка	DevOps-інженер	$30-50 за годину
Хостинг або сервери	AWS S3, GitLab Pages	$0,023 за Гб
Навчання співробітників	Курси або внутрішні тренінги	$500-1000 за тренінг на команду

Аналіз витрат на впровадження автоматизації документування дозволяє зробити висновок, що основні фінансові вкладення пов'язані з ліцензуванням програмного забезпечення, налаштуванням інфраструктури та навчанням команди.

Однак, у довгостроковій перспективі ці витрати компенсуються завдяки зниженню витрат на ручну працю, підвищенню продуктивності команди та мінімізації помилок у документації. Використання безкоштовних інструментів, таких як GitLab Pages або AWS S3 для хостингу, дозволяє оптимізувати витрати, що особливо важливо для малих і середніх проєктів.

3.4.2 Оцінка економічної ефективності

Процес впровадження автоматизації документування в IT-проєктах описує діаграма послідовності, що демонструє взаємодію між менеджером проєкту, інженером з автоматизації, інструментами автоматизації та системою документування (рисунок 3.4).

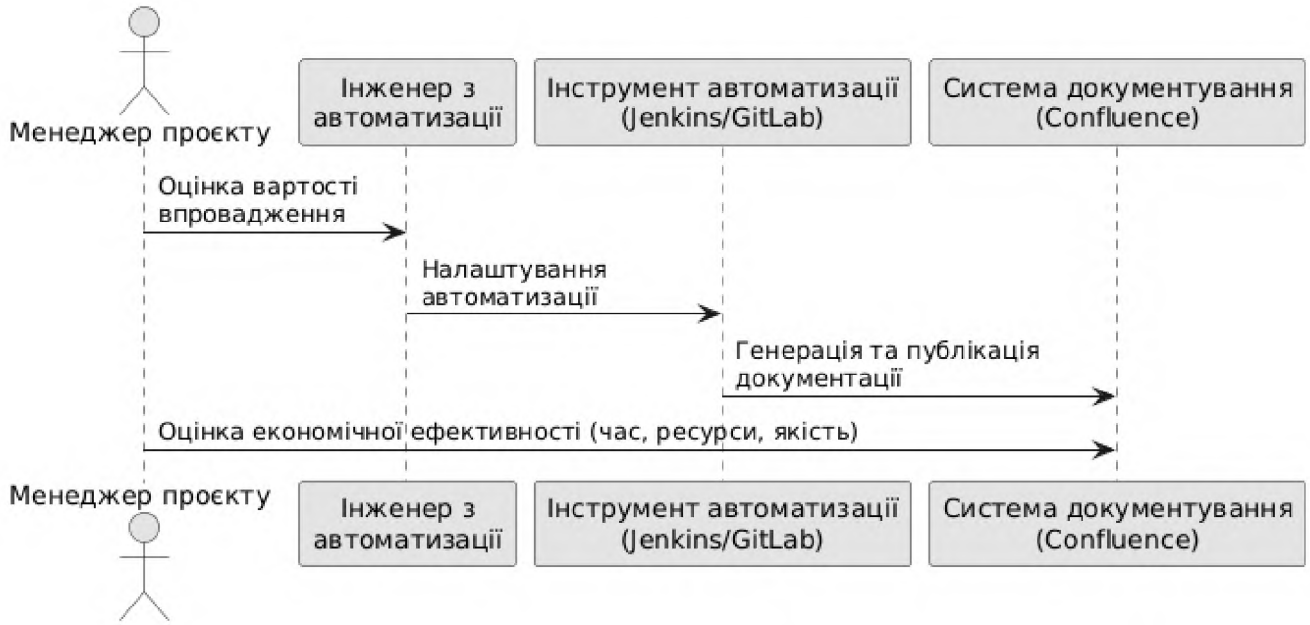

Рисунок 3.4 – Процес впровадження автоматизації документування в IT-проєктах

Дана діаграма послідовності відображає процес впровадження автоматизації документування в IT-проєктах:

- менеджер проєкту ініціює процес, оцінюючи вартість впровадження автоматизації та передаючи необхідні дані інженеру з автоматизації. Цей етап передбачає аналіз витрат на інструменти, ресурси та очікувані результати;

- інженер з автоматизації виконує налаштування автоматизації за допомогою інструментів, таких як Jenkins або GitLab CI/CD. Цей етап включає налаштування пайплайнів, інтеграцію систем і вибір інструментів для документування;

- інструмент автоматизації генерує та публікує документацію у системі документування, наприклад, у Confluence. Це відбувається автоматично після кожної зміни в проєкті або на основі заданих тригерів;

- менеджер проєкту оцінює економічну ефективність впровадження автоматизації, аналізуючи ключові показники: час, ресурси та якість створеної документації.

Для оцінки економічної ефективності впровадження автоматизації документування в IT-проєктах розглянемо вплив автоматизації на час та ресурси.

Автоматизація знижує час, необхідний для створення і оновлення документації. Дійсно, ручне документування займає кілька годин на день, і при цьому дуже часто документація відстає від коду. За допомогою автоматизованих рішень, таких як Jenkins та GitLab CI, процес оновлення документації може займати лише декілька хвилин після кожної зміни в коді.

Автоматизація документування дозволяє зменшити кількість залучених до процесу працівників. Наприклад, замість того, щоб залучати кількох інженерів до підтримки документації, можна автоматизувати цей процес, зберігши ресурси.

Приклад порівняння витрат ресурсів на документування до та після автоматизації представлене у таблиці 3.6.

Таблиця 3.6 – Витрати ресурсів на документування до та після автоматизації

Види ресурсів	До автоматизації	Після автоматизації
Людські ресурси	2-3 співробітника	1 співробітник
Час на створення документації	5 годин на місяць	30 хвилин
Грошові витрати на підтримку документації	Високі	Низькі

Техніко-економічне обґрунтування впровадження автоматизації документування показує, що автоматизація не тільки знижує витрати на ручну роботу і підтримку документації, але також суттєво підвищує її якість та актуальність. Інвестиції в інструменти та навчання можуть швидко окупитися завдяки економії часу, покращенню управління ресурсами та зменшенню ризику помилок у проєктах.

Для виконання розрахунку ROI (Return on Investment) та терміну окупності інвестицій в автоматизацію документування необхідно визначити критерії для малих, середніх та великих проєктів, розробити докладний кошторис витрат, а також оцінити вигоди від впровадження автоматизації.

Критерій для оцінки розміру IT-проєктів представлені у таблиці 3.7.

Таблиця 3.7 – Критерії для оцінки розміру IT-проєктів

Розмір проєкту	Кількість розробників	Обсяг документації	Час на документування (до автоматизації)	Приклади проєктів
Малий	До 5	До 50 сторінок	10-15 годин на тиждень	Невеликі вебсайти, MVP-продукти
Середній	6-20	50-200 сторінок	20-40 годин на тиждень	Корпоративні системи, API-платформи
Великий	20+	Понад 200 сторінок	40+ годин на тиждень	Великі IT-системи, комплексні рішення

Розглянемо складові кошторису витрат на автоматизацію документування IT-проєктів. Основні статті витрат включають:

- витрати на ліцензії на інструменти – платні рішення для генерації та публікації документації (GitLab CI, Jenkins, Confluence, Swagger);

- витрати на інфраструктуру – налаштування CI/CD, хостинг документації на платформах GitHub Pages, AWS або іншому хмарному середовищі;

- витрати на навчання персоналу (розробників та технічних письменників);

- витрати на інтеграцію – налаштування автоматизованих пайплайнів для генерації документації;

- витрати на підтримку та оновлення – щорічна підтримка та доопрацювання автоматизованих рішень.

Примірний кошторис витрат на автоматизацію документування малих, середніх та великих IT-проєктів представлений у таблиці 3.8.

Таблиця 3.8 – Витрати на автоматизацію документування IT-проєктів

Розмір проєкту	Ліцензії на інструменти (USD)	Інфраструктура (USD)	Навчання персоналу (USD)	Інтеграція (USD)	Підтримка (USD/рік)	Загальні витрати (USD)
Малий	500	300	800	1000	300	2900
Середній	1500	1000	1500	2500	1000	7500
Великий	5000	3000	3000	5000	2500	18500

Очікувані вигоди від автоматизації документування:

- економія часу на створення, оновлення та підтримку документації;
- зменшення кількості помилок;
- покращення якості – документація завжди актуальна та відповідає коду.

Грошова оцінка вигоди за рахунок автоматизації документування малих, середніх та великих IT-проєктів представлена у таблиці 3.9.

Таблиця 3.8 – Вигоди за рахунок автоматизації документування IT-проєктів

Розмір проєкту	Час на документування до автоматизації	Час після автоматизації	Зекономлений час (год/місяць)	Економія (USD/рік, $20/год)
Малий	60 годин на місяць	20 годин на місяць	40 годин	9600
Середній	160 годин на місяць	50 годин на місяць	110 годин	26400
Великий	400 годин на місяць	120 годин на місяць	280 годин	67200

На основі отриманих даних, виконаємо розрахунок коефіцієнту повернення інвестицій (ROI) та терміну окупності інвестицій у впровадження автоматизації IT-проєктів.

Формула для розрахунку ROI:

$$ROI = \frac{Вигоди - Витрати}{Витрати} \times 100\%. \quad (3.1)$$

Формула для розрахунку терміну окупності:

$$T = \frac{Витрати}{Вигоди}. \quad (3.2)$$

Для малих проєктів за формулами (3.1) і (3.2) обчислимо:

$$ROI = \frac{9600 - 2900}{9600} \times 100\% \approx 231\%,\ T = \frac{2900}{9600} \approx 0,302 \approx 3,6\ міс.$$

Для середніх проєктів:

$$ROI = \frac{26400 - 7500}{26400} \times 100\% \approx 252\%,\ T = \frac{7500}{26400} \approx 0,284 \approx 3,4\ міс.$$

Для великих проєктів:

$$ROI = \frac{67200 - 18500}{67200} \times 100\% \approx 263\%,\ T = \frac{18500}{67200} \approx 0,275 \approx 3,3\ міс.$$

Таким чином, впровадження автоматизації документування в IT-проєктах є високоефективним інвестиційним рішенням. Усі категорії проєктів (малі, середні, великі) демонструють швидкий термін окупності (3-4 місяці) та високий ROI (від 231% до 263%). Найбільший ефект досягається у великих проєктах, де економія часу і зниження витрат найбільш відчутні.

Висновки до розділу 3

Модель автоматизації документування включає кілька головних компонентів: систему керування версіями, CI/CD пайплайни для автоматичного збирання документації, інструменти для генерації документації та платформи для її публікації. Така архітектура дозволяє автоматизувати процеси створення та оновлення документації, що підвищує ефективність проєктів та зменшує кількість помилок.

Програмна реалізація автоматизації документування IT-проєктів базується на інтеграції систем керування версіями, CI/CD інструментів та систем для

генерації і публікації документації. За допомогою таких інструментів, як Jenkins, GitLab CI, Sphinx, MkDocs, та Confluence, за допомогою яких можна створити повністю автоматизовану систему для підтримки актуальної документації. Інтеграція цих інструментів забезпечує високу швидкість та точність, знижуючи кількість помилок та ручної роботи.

Впровадження автоматизації документування в IT-проєктах вимагає підходу з урахуванням існуючих процесів, правильного вибору інструментів та вирішення потенційних технічних труднощів. Завдяки правильній інтеграції систем CI/CD, таких як GitLab або Jenkins, із системами для генерації документації, можна забезпечити постійну актуальність документації, мінімізувати кількість помилок та значно зменшити витрати часу на підтримку документації.

Економічна оцінка ефективності показує, що автоматизація IT-проєктів не тільки знижує витрати на ручну роботу і підтримку документації, але також підвищує її якість та актуальність. Інвестиції в інструменти та навчання, пов'язані з впровадженням автоматизації документування, можуть швидко окупитися завдяки економії часу, покращенню управління ресурсами та зменшенню ризику помилок у проєктах.

ВИСНОВКИ

У результаті виконаної роботи було показано, що автоматизація процесів документування є важливим елементом управління IT-проєктами, який значно підвищує ефективність процесу документування та якість документації. Аналіз сучасних підходів показав, що використання інструментів автоматизації, таких як GitLab CI, Jenkins, Swagger, Sphinx та інших для автоматизації документування IT-проєктів дозволяє мінімізувати ручну працю, знизити кількість помилок у документації, забезпечити її актуальність і відповідність реальному стану проєкту.

Дослідницько-аналітичний аспект роботи виявив, що інтеграція інструментів автоматизації з системами контролю версій і CI/CD процесами допомагає не тільки економити час, але й підвищує точність документування. Впровадження таких рішень в реальні проєкти створює позитивний вплив на зниження витрат, прискорення процесу створення документації та покращення загального управління проєктами.

На основі запропонованої моделі автоматизації, яка включає систему керування версіями, інструменти для CI/CD і генерації документації, було показано, що автоматизація може бути успішно впроваджена в різні проєкти. Програмна реалізація автоматизації документування, що базується на таких інструментах, як Sphinx, MkDocs, GitLab Pages та Confluence, цілком забезпечує безперервний процес оновлення і публікації документації, що гарантує її відповідність змінам у коді та специфікаціях.

Результати техніко-економічного аналізу дозволяють зробити висновок, що автоматизація не лише знижує витрати на підтримку документації, але також суттєво покращує її якість, точність і актуальність. Незначні інвестиції в навчання команди та впровадження інструментів швидко окупаються завдяки зниженню кількості помилок і економії часу.

Перспективи подальших досліджень у сфері автоматизації документування полягають у розвитку інтеграції інструментів штучного інтелекту (ШІ) для

автоматичного аналізу коду і генерації документації, вдосконаленні алгоритмів аналізу природної мови для створення більш деталізованих і зрозумілих описів, а також розширенні можливостей інтеграції автоматизації з різними системами управління проєктами і хмарними рішеннями. Крім того, важливими напрямами залишаються розробка інструментів для мультиплатформеної підтримки та стандартизації процесів документування з урахуванням специфіки різних мов програмування і методологій розробки.

Елементи новизни даної роботи полягають у розробці комплексної моделі автоматизації процесу документування IT-проєктів, яка включає інтеграцію сучасних інструментів для CI/CD, генерації та публікації документації. Також новизна полягає в застосуванні обґрунтованого підходу до впровадження автоматизації з акцентом на зниження кількості помилок, підвищення швидкості оновлення документації та зменшення витрат на її підтримку.

Результати роботи можуть бути використані на практиці для впровадження автоматизованих систем документування в реальних IT-проєктах, що дозволить суттєво підвищити ефективність управління проєктами. Запропонована модель автоматизації може бути адаптована до різних середовищ розробки, що забезпечить актуальність і точність документації на всіх етапах життєвого циклу проєкту.