МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 07.00.00.000 ПЗ

Група ІПм-24-2

Івасишин Роман

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Івасишин Роман Васильович

(прізвище, ім'я, по батькові)

УДК 004.9

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи систем планування та відстеження

програмних проектів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Івасишин Р.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Вовк Роман Богданович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**

доц. В.В. Бандура

“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Івасишину Роману Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи систем планування та відстеження програмних проектів”

керівник проекту (роботи) Вовк Роман Богданович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проекту (роботи) 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій планування IT проектів

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження предметної області використання гнучких методологій
2. Методології та методи планування програмних проектів
3. Імплементація методів та методологій в системі планування та відстеження проектів
4. Проектування системи управління проектами

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Представлення процесу дослідження (рис. 1.1)
2. Графічне представлення Dynamic System Development Method (рис. 1.2)
3. Процеси в Crystal методології (рис. 1.3)
4. Структура методології ASD (рис. 1.4)
5. Структура FDD методології (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2025	виконано
2	Дослідження предметної області використання гнучких методологій	29.09.2025	виконано
3	Методології та методи планування програмних проектів	15.10.2025	виконано
4	Імплементація методів та методологій в системі планування та відстеження проектів	08.11.2025	виконано
5	Проектування системи управління проектами	20.11.2025	виконано
6	Реалізація функціональності запропонованої інформаційної технології	01.12.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 76 с., 29 рис., 2 табл., 39 джерел.

Тема: Моделі та методи систем планування та відстеження програмних проектів

Мета роботи - розробка моделей та методів систем планування й відстеження програмних проєктів, що базуються на інтеграції гнучких методологій, та створення прототипу системи управління проєктами.

Об'єкт дослідження - процес управління програмними проєктами із застосуванням гнучких методологій розробки програмного забезпечення.

Предмет дослідження - моделі, методи та засоби планування й відстеження програмних проєктів на основі інтеграції Scrum та Extreme Programming.

Результати дослідження

Запропоноване рішення дозволяє підвищити рівень автоматизації процесів планування та відстеження, забезпечити своєчасне реагування на зміни вимог, а також створює передумови для подальшої масштабованості та розширення функціональності системи

Висновок

Практична цінність роботи полягає у можливості впровадження запропонованих рішень у діяльність ІТ-команд, що дозволить підвищити продуктивність розробки, скоротити ризики невідповідності вимогам замовників та оптимізувати управління ресурсами.

УПРАВЛІННЯ ПРОГРАМНИМИ ПРОЄКТАМИ, ГНУЧКІ МЕТОДОЛОГІЇ, SCRUM, EXTREME PROGRAMMING, AGILE, ПЛАНУВАННЯ, ВІДСТЕЖЕННЯ, МОДЕЛІ, МЕТОДИ, СИСТЕМНА МОДЕЛЬ, КРИТЕРІЇ ОЦІНКИ, ПРОТОТИП СИСТЕМИ.

ABSTRACT

Master Thesis: 76 pp., 29 fig., 2 tab., 39 sources.

Topic: Models and methods of software project planning and tracking systems

The purpose of the work is to develop models and methods of system planning and tracking of software projects based on the integration of agile methodologies, and to create a prototype of a project management system.

The object of the study is the process of software project management using agile software development methodologies.

The subject of the study is models, methods and tools for planning and tracking software projects based on the integration of Scrum and Extreme Programming.

Research results

The proposed solution allows to increase the level of automation of planning and tracking processes, to ensure timely response to changes, and to create prerequisites for further scalability and expansion of the system's functionality.

Conclusion

The practical value of the work will arise from the possibility of implementing the proposed solutions in the activities of IT teams, which allow to increase productivity. development, reduce the risks of non-compliance with customer requirements and optimize resource management.

SOFTWARE PROJECT MANAGEMENT, FLEXIBLE METHODOLOGIES, SCRUM, EXTREME PROGRAMMING, AGILE, PLANNING, TRACKING, MODELS, METHODS, SYSTEM MODEL, EVALUATION CRITERIA, SYSTEM PROTOTYPE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СППП - система планування програмного проекту

СПВПП - Система планування і відстеження програмних проектів

PPTS - Project Planning and Tracking System

XP - Extreme Programming

DSDM - Dynamic System Development Method

ВСТУП

Актуальність теми.

Сучасна індустрія розробки програмного забезпечення характеризується високою динамікою змін, жорсткою конкуренцією та необхідністю швидкого реагування на потреби замовників. Це зумовлює потребу у вдосконаленні систем управління програмними проєктами, які забезпечують ефективне планування, контроль виконання завдань та адаптацію до змін. Традиційні каскадні підходи дедалі частіше поступаються місцем гнучким методологіям, таким як Scrum, Extreme Programming (XP), Kanban та їхні гібридні комбінації, що дозволяють підвищити якість продукту, мінімізувати ризики та забезпечити безперервну комунікацію в команді.

Попри значний прогрес у розвитку інструментів управління проєктами, існує низка невирішених проблем, пов’язаних із комплексною інтеграцією моделей планування та відстеження, зручністю використання систем, а також об’єктивністю критеріїв оцінювання ефективності таких рішень. Це визначає необхідність проведення дослідження моделей і методів систем планування та відстеження програмних проєктів, що мають практичне значення для IT-індустрії та можуть сприяти підвищенню продуктивності командної роботи.

Актуальність дослідження зумовлена зростанням ролі програмних проєктів у цифровій економіці та потребою у високоефективних інструментах їх управління. Гнучкі методології (Agile, Scrum, XP) довели свою ефективність у швидкоплинних умовах розробки, проте їхня імплементація потребує удосконалених моделей інтеграції та адаптації. Важливим завданням є розробка систем, що поєднують планування, відстеження й аналітику, забезпечуючи повну прозорість процесів і зменшення ризиків невідповідності кінцевого продукту вимогам замовника.

Отже, робота є актуальною, оскільки спрямована на вирішення завдань підвищення ефективності управління програмними проєктами шляхом

поєднання теоретичних моделей, практичних методів і засобів їхньої імплементації у вигляді інтегрованої системи планування та відстеження.

Метою роботи є розробка моделей та методів систем планування й відстеження програмних проєктів, що базуються на інтеграції гнучких методологій, та створення прототипу системи управління проєктами.

Об'єкт дослідження - процес управління програмними проєктами із застосуванням гнучких методологій розробки програмного забезпечення.

Предмет дослідження - моделі, методи та засоби планування й відстеження програмних проєктів на основі інтеграції Scrum та Extreme Programming.

Завдання дослідження

1. Провести аналіз предметної області використання гнучких методологій управління програмними проєктами.

2. Дослідити ключові принципи та інструменти Scrum і Extreme Programming, визначити можливості їх інтеграції.

3. Розробити критерії оцінювання ефективності систем планування й відстеження програмних проєктів.

4. Сформувати системну модель управління програмними проєктами з урахуванням ієрархії робочих елементів.

5. Створити прототип системи управління програмними проєктами, що реалізує основні функції планування та контролю.

Методи дослідження

У роботі використано системний підхід до аналізу предметної області, методи порівняльного аналізу та узагальнення, моделювання бізнес-процесів і життєвого циклу програмних продуктів. Для перевірки працездатності розроблених рішень застосовано методи експериментальної перевірки на основі тестових даних і структурованих сценаріїв планування.

Наукова новизна отриманих результатів

Запропоновано інтегровану модель планування та відстеження програмних проєктів, що поєднує практики Scrum та XP. Розроблено критерії

оцінювання ефективності систем управління проєктами з урахуванням прозорості, гнучкості та зручності використання. Імплементовано прототип системи управління проєктами з оптимізованим інтерфейсом менеджера, що забезпечує підвищення продуктивності роботи команди.

Практичне застосування результатів

Розроблені моделі та методи можуть бути використані у практичній діяльності IT-компаній для вдосконалення процесів управління програмними проєктами. Запропонована система дозволяє підвищити ефективність планування, скоротити ризики невідповідності вимогам, оптимізувати використання ресурсів та поліпшити комунікацію в командах. Результати дослідження можуть бути інтегровані як у корпоративні рішення, так і в системи з відкритим кодом для управління проєктами.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів, висновків та додатків. Загальний обсяг роботи становить 76 сторінок, і містить 29 рисунків, 2 таблиці, список використаних джерел із 39 найменувань.

РОЗДІЛ 1. ОГЛЯД ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ВИКОРИСТАННЯ ГНУЧКИХ МЕТОДОЛОГІЙ ДЛЯ ПЛАНУВАННЯ ТА ВІДСТЕЖЕННЯ ПРОГРАМНИХ ПРОЕКТІВ

1.1. Застосування гібридних гнучких методологій в розробці програмного забезпечення

IT компанії активно застосовують методології гнучкої розробки, зокрема XP@Scrum. Цей підхід є комбінацією двох ітеративних та інкрементальних методологій: Extreme Programming (XP) та Scrum. Обидві методології зосереджені на роботі невеликих команд, які в кінці кожної ітерації створюють потенційно готове до випуску програмне забезпечення.

XP@Scrum базується на ітеративній та інкрементальній розробці, що є центральною ідеєю як Scrum, так і XP. У цьому підході процеси організовані наступним чином:

Scrum-складова забезпечує організаційний фреймворк. Вона використовується для планування, відстеження прогресу та управління проектом. Це включає в себе такі елементи, як спринти (ітерації тривалістю 2–4 тижні), планування спринтів, щоденні стендапи (daily scrums), огляди спринтів (sprint reviews) та ретроспективи (retrospectives). Завдяки цьому команди можуть швидко адаптуватися до змінних вимог і підтримувати прозорість процесу.

XP-складова зосереджена на технічних аспектах розробки та підвищенні якості коду. Вона включає такі практики:

- Розробка через тестування (TDD): написання автоматизованих тестів до написання коду. Це гарантує, що код буде працювати правильно, і спрощує майбутній рефакторинг.

- Парне програмування (Pair Programming): два розробники працюють разом за одним комп'ютером. Це покращує якість коду, прискорює обмін знаннями та зменшує кількість помилок.

- Безперервна інтеграція (Continuous Integration): часті інтеграції коду в загальну кодову базу, що допомагає швидко виявляти та виправляти конфлікти.

- Рефакторинг (Refactoring): постійне покращення структури коду без зміни його зовнішньої поведінки. Це підтримує код чистим і легким для розуміння та подальшої модифікації.

Об'єднання цих методологій дозволяє створювати високоякісне програмне забезпечення з більшою гнучкістю, що зрештою призводить до підвищення задоволеності клієнтів. XP@Scrum є ефективним рішенням для команд, які прагнуть поєднати організаційну ефективність Scrum з високими технічними стандартами XP.

1.1.1. Аналіз та конкурентний огляд системи планування програмного проекту

Для підтримки команд IT компанії часто розробляють власні системи планування програмного проекту (СППП) — веб-інструменту для планування та відстеження проектів. Однак за останні роки на ринку з'явилося безліч альтернативних рішень, як комерційних, так і з відкритим кодом. Компанії прагнуть оцінити позиції СППП порівняно з цими конкурентами. Результати цього порівняння послугують основою для розробки нової версії СППП, яка виправить наявні недоліки та додасть нову функціональність. Особлива увага приділяється аспектам зручності використання (usability), оскільки поточний інтерфейс далекий від оптимального. Основна методологія, яку підтримує СППП, — це Scrum, і IT компанія має на меті зробити СППП основним інструментом для розробників, що працюють за цією методологією.

Метою даного дослідження є оцінка СППП та її конкурентів, а також аналіз їхніх відмінностей та схожих рис. На основі цього аналізу будуть сформульовані рекомендації щодо функцій, які варто інтегрувати в СППП, та змін, необхідних для покращення зручності використання поточних функцій.

Ключовий акцент робиться на зручності використання функцій планування та інтерфейсу користувача, які потребують суттєвого вдосконалення. Додатковою метою є порівняння розглянутих інструментів для загального використання в Agile-спільноті, щоб надати рекомендації щодо вибору найкращого інструменту для конкретного проекту.

Соціальна значущість цього дослідження полягає в тому, що IT компанія потребує обґрунтованих даних для внесення змін до інструменту СППП. Наразі деякі команди не використовують цей інструмент через проблеми з зручністю, і компанія прагне змінити цю ситуацію. Першим кроком є визначення всіх можливих варіантів покращення.

Наукова значущість цього дослідження полягає у відсутності детального аналізу інструментів для Agile-розробки на момент його проведення. З огляду на існуючий попит з боку консультантів з Agile-методологій та інших дослідників у цій галузі, ця робота стане цінним джерелом інформації.

1.1.2. Методологія дослідження та аналізу

Спочатку буде проведено детальний аналіз систем планування та відстеження проектів. Метою цього аналізу є виявлення функцій, які потребують впровадження або модифікації в системі. Для забезпечення максимальної ефективності дослідження буде проведено вивчення загальних методологій Agile, а також розроблено набір тестових даних для об'єктивного порівняння різних інструментів.

Використовуючи розроблений набір тестових даних, буде проведена оцінка різних інструментів. При цьому будуть задокументовані їхні сильні та слабкі сторони. Система також буде протестована і слугуватиме еталонним зразком для порівняння.

Виявлені переваги інших інструментів у поєднанні з недоліками системи планування проектів стануть основою для формування переліку потенційних змін.

Особлива увага в рамках аналізу буде приділена аспектам зручності використання (usability) конкурентних інструментів, особливо в контексті функцій планування. Ця область є критичною для вдосконалення системи. Візуальне представлення запланованого процесу дослідження наведено на рисунку 1.1.

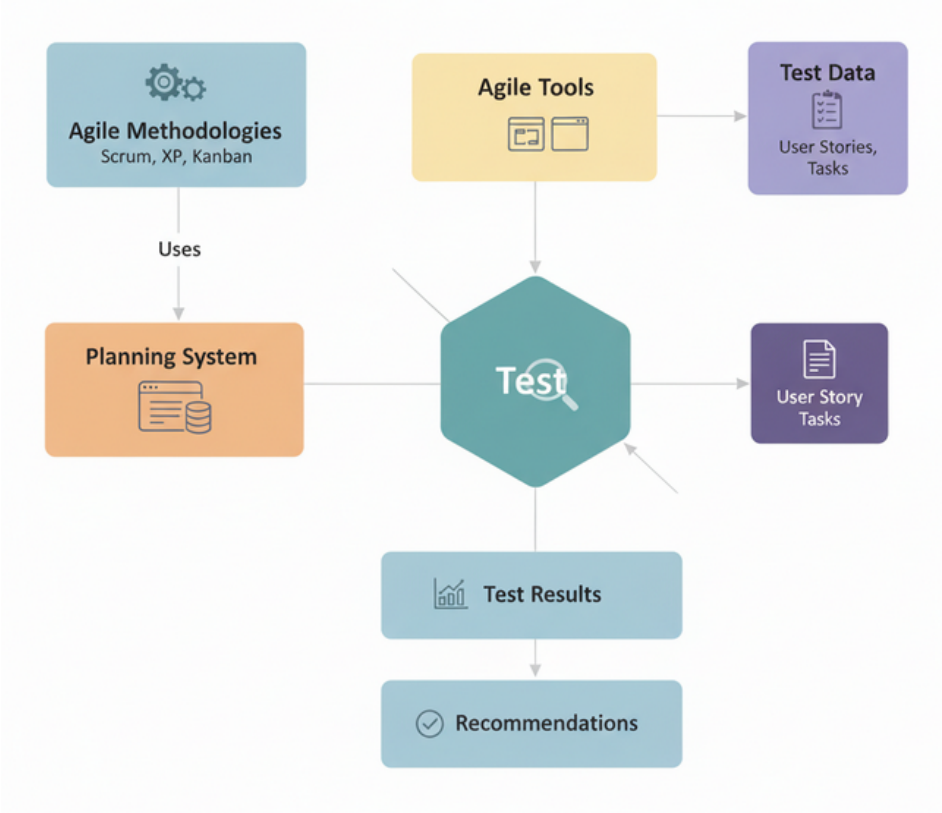

Рис. 1.1. Представлення процесу дослідження

1.2. Формування парадигми гнучкої розробки програмного забезпечення: історія та принципи

Гнучка розробка програмного забезпечення охоплює широкий спектр методологій, кожна з яких має свою унікальну специфіку та набір ключових практик. Попри відмінності, всі вони поділяють спільні принципи, що формують їхню сутність та визначають їх як "гнучкі".

Ідеї, що лежать в основі гнучких методологій, існували задовго до появи самого терміна. Рання форма цього підходу відома як ітеративна та

інкрементальна розробка (IID). Одним із перших прикладів успішного застосування IID у розробці програмного забезпечення є проект NASA Mercury (1960-ті роки). У цьому проекті, який призвів до першого пілотованого космічного польоту США, програмне забезпечення створювалося ітераціями, з використанням практики тестування перед розробкою. Цей підхід, що передбачає написання тестів до написання коду, є однією з фундаментальних практик сучасних гнучких методологій. Успіх цього підходу згодом був підтверджений в IBM Federal Systems Division (FSD), де колишні члени команди Mercury реалізували низку успішних IID-проектів для Міністерства оборони.

У 1990-х роках IID привернула широку увагу, що призвело до її кодифікації у формальні методології гнучкої розробки. Серед найвідоміших методологій, що розвивалися паралельно, можна виділити Scrum, Extreme Programming (XP) та Dynamic Systems Development Method (DSDM).

Scrum був вперше застосований у 1993 році і формалізований як спроба пояснити високу продуктивність у певних об'єктно-орієнтованих проектах. [2]. Його розробка ґрунтувалася на аналізі процесів успішних незалежних постачальників програмного забезпечення (ISV) з метою ідентифікації та поширення ефективних практик.

Extreme Programming розвинувся в проекті Chrysler C3 [4]. XP вважається однією з найбільш "екстремальних" форм гнучких методологій, оскільки зосереджена майже виключно на інженерних практиках. Цей підхід отримав як прихильників, так і критиків, які іноді називають його "формалізованим хакінгом".

DSDM став результатом колективних зусиль 16 розробників, які прагнули створити стандартний ітеративний процес для підтримки швидкої розробки додатків (RAD). DSDM включав елементи, такі як таймбоксинг.

Гнучка розробка програмного забезпечення не є універсальним рішенням, проте вона має значні переваги в середовищах зі швидкозмінними вимогами, оскільки підтримує адаптацію до змін.

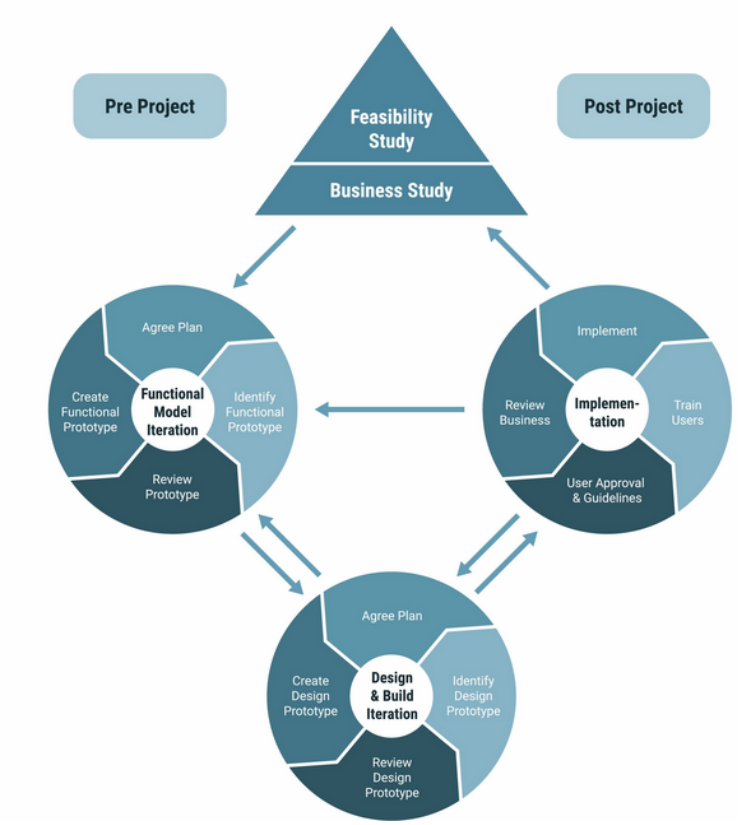

Рис. 1.2. Графічне представлення Dynamic System Development Method

Діаграма (рис. 1.2) ілюструє, як DSDM організовує проект через етапи: передітераційна, ітераційна та постітераційна. Вона наголошує на таких ключових аспектах, як бізнес-вимоги, розробка, тестування та розгортання, показуючи циклічний і спільний характер методології.

Філософія гнучкої розробки програмного забезпечення була формалізована в Маніфесті Agile, створеному в 2001 році на зустрічі 17 провідних розробників. Маніфест визначає чотири ключові цінності, що відрізняють гнучкий підхід від традиційних:

- Люди та взаємодія цінуються більше, ніж процеси та інструменти.

- Працююче програмне забезпечення цінується більше, ніж вичерпна документація.

- Співпраця з замовником цінується більше, ніж узгодження умов контракту.

- Реагування на зміни цінується більше, ніж дотримання плану.

Ці принципи, хоча й визнають цінність елементів праворуч, надають пріоритет елементам ліворуч. Спочатку ці методології називалися "легкими" (lightweight), але через невдоволення цією назвою, був прийнятий термін "гнучкі" (Agile). Маніфест Agile змістив фокус від жорстких, бюрократичних процесів до гнучкості та взаємодії. Це також розглядається як передача більшої влади від менеджменту до розробників, що є очікуваним, оскільки більшість авторів маніфесту були розробниками. Проте, наукових даних про успішність гнучких проектів порівняно з традиційними методологіями, такими як каскадна модель, наразі недостатньо, хоча існують дослідження окремих практик, наприклад, парного програмування [5].

1.3. Ключова термінологія Agile-розробки

Перед розглядом конкретних методологій необхідно визначити низку ключових термінів, що є основою гнучкої розробки програмного забезпечення.

1.3.1. Елементи планування та оцінювання

Історія користувача (User Story) - опис необхідної функціональності системи, викладений з позиції кінцевого користувача. Це стислий, неформальний опис вимоги, що надається замовником і формулюється в декількох реченнях. Історії користувачів представляють функціональні можливості різного масштабу, які інтегруються в запланований програмний продукт.

Завдання (Task) - Декомпозиція історії користувача на дрібніші, конкретні кроки, необхідні для її реалізації. Завдання визначаються командою розробників і зазвичай включають кодування та супутні дії.

Ідеальні дні/години (Ideal Days/Hours) - абстрактна одиниця часу, що відображає тривалість роботи розробника без перерв і відволікань. Ця

одиниця часто використовується для оцінювання обсягу історій користувача та завдань.

Оцінка в балах (Story Points) - Довільна одиниця вимірювання, що оцінює відносний обсяг роботи. Наприклад, обсяг роботи на 2 бали вважається вдвічі більшим, ніж на 1 бал. Перевага використання балів полягає у відсутності прямого зв'язку з часом, на відміну від "ідеальних годин".

Коефіцієнт навантаження (Load Factor) - множник, що використовується для перетворення "ідеального" часу на "реальний" робочий час. Цей показник враховує непроектну діяльність (наприклад, зустрічі, адміністративні завдання, перерви). Типові значення коефіцієнта навантаження варіюються від 1,3 до 1,6, що вказує на те, що 30-60% робочого часу присвячено непрямим витратам.

Швидкість (Velocity) - еквівалент коефіцієнта навантаження в Scrum при використанні оцінок у балах. Цей термін вказує на кількість балів, виконаних командою протягом однієї ітерації. Швидкість також може виражатися у відсотках, що відображає частку "ідеального" часу у загальному доступному часі. Наприклад, швидкість 70% означає, що 70% часу було витрачено безпосередньо на виконання завдань проекту, а 30% — на накладні витрати.

1.3.2. Технічні практики та терміни

Одиниця (Unit) - найменший, незалежно компонент тестування програмного коду, такий як функція або процедура.

Одиничний тест (Unit Test) - автоматизований тест, що перевіряє поведінку окремої одиниці, подаючи на неї вхідні дані та порівнюючи отримані результати з очікуваними.

Рефакторинг (Refactoring) - це внесення змін у внутрішню структуру програмного забезпечення з метою полегшення його розуміння та зниження вартості модифікації, при цьому не змінюючи його зовнішньої поведінки.

Рефакторинг допомагає підтримувати код зрозумілим і гнучким для подальшого розвитку проекту.

1.4. Огляд та порівняльний аналіз основних гнучких методологій

У попередніх розділах було зазначено, що, попри спільні об'єднуючі принципи, існують різні гнучкі методології, кожна з яких має унікальний набір практик. Нижче наведено стислий опис деяких із найбільш відомих підходів до гнучкої розробки програмного забезпечення.

Extreme Programming (XP) — методологія, орієнтована на інтенсивне використання інженерних практик для підвищення якості коду та гнучкості.

Crystal — сімейство методологій, розроблених для проектів різного масштабу та важливості. Назви варіюються від Crystal Clear (для малих проектів) до Crystal Red (для великих проектів). Спільними для цих підходів є інкрементальне постачання, відстеження прогресу за допомогою віх, активна участь замовника, автоматизоване регресійне тестування, регулярні огляди та майстер-класи.

Рис.1.3. Процеси в Crystal методології

Ця діаграма (рис. 1.3) демонструє сімейство методологій Crystal з акцентом на їхній масштабованості. Вона візуально показує, як різні версії методології (Clear, Yellow, Orange, Red) адаптуються до розміру команди та критичності проекту.

Scrum — методологія, що фокусується на управлінні проектами через ітеративні цикли (спринти), що забезпечує гнучкість та прозорість процесу.

Adaptive Software Development (ASD) — підхід, що спрямований на управління великими та складними проектами. ASD виступає за ітеративну розробку з акцентом на прототипування, прагнучи забезпечити стабільність без пригнічення креативності та емерджентності.

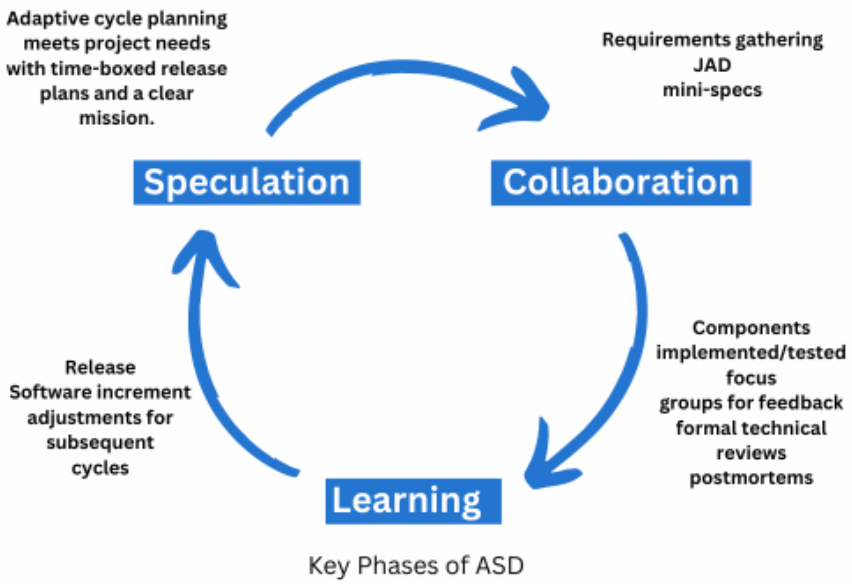

Рис. 1.4. Структура методології ASD

Методологія Adaptive Software Development була створена для заміни традиційної моделі водоспаду. Вона є гнучким підходом, орієнтованим на постійні зміни та навчання. ASD складається з трьох ключових фаз: Спекуляція, Співпраця та Навчання.

Спекуляція (Speculation) - замість традиційного детального планування на початку, команда "спекулює" про можливі майбутні результати, адаптуючись до невизначеності.

Співпраця (Collaboration) - команди працюють разом над розробкою продукту, використовуючи спільні знання та навички для вирішення проблем.

Навчання (Learning) - після завершення кожного циклу, команда аналізує результати та виносить уроки, що допомагає вдосконалювати подальші процеси.

ASD підкреслює важливість адаптації, швидкого реагування на зміни та безперервного навчання, що робить її ефективною в нестабільному та динамічному середовищі.

Feature Driven Development (FDD) — методологія, що охоплює фази проектування та побудови, акцентуючи увагу на якості, моніторингу процесу та частому, чітко визначеному постачанні функцій. FDD може інтегруватися з іншими практиками розробки.

FDD — це ітеративна та інкрементальна методологія розробки програмного забезпечення, що належить до родини гнучких (Agile) підходів. FDD зосереджується на створенні "функцій" (features) — невеликих, цінніснх для користувача елементів функціональності. Кожна функція має чітко визначену форму: "<дія> <результат> <об'єкт>" (наприклад, "обробка платежу клієнта"). Цей підхід дозволяє швидко досягати прогресу та забезпечувати видимі результати.

Методологія FDD включає п'ять основних процесів, які виконуються ітеративно:

1. Розробка загальної моделі (Develop an Overall Model).

На цьому етапі створюється загальна модель системи, що включає об'єкти, їх взаємозв'язки та функції. Ця модель служить базою для подальшої розробки.

2. Формування списку функцій (Build a Features List).

На основі моделі та вимог замовника створюється ієрархічний список функцій, які необхідно реалізувати. Функції групуються за областями та підрозділами.

3. Планування за функціями (Plan by Feature).

Визначається порядок реалізації функцій, розподіляються ресурси та формується план роботи.

4. Проектування за функціями (Design by Feature).

Кожна функція детально проектується, створюються діаграми послідовності та інші проектні артефакти.

5. Створення за функціями (Build by Feature).

Це етап безпосередньої розробки, коли код пишеться, тестується та інтегрується.

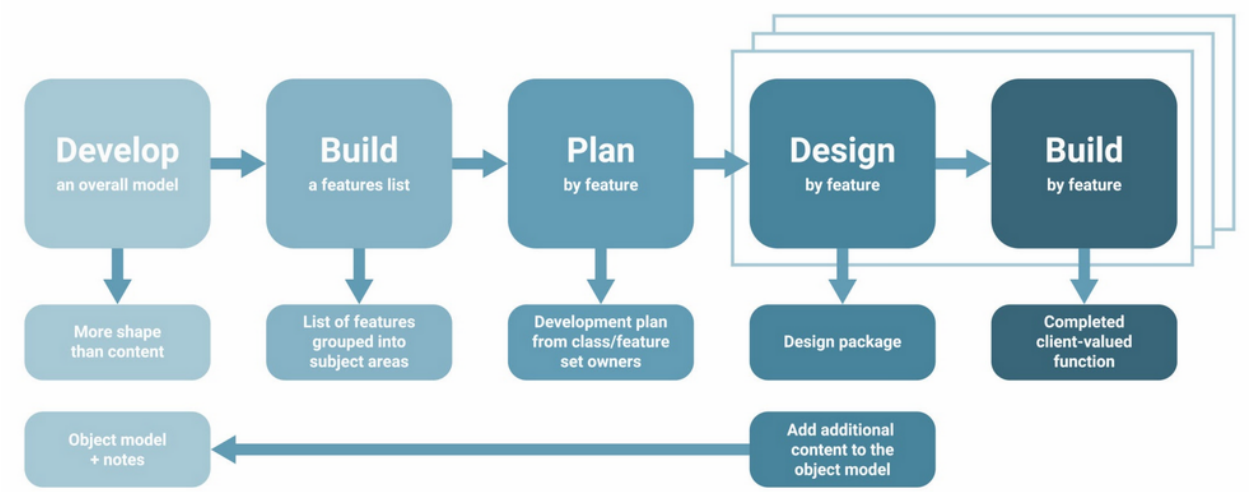

Рис. 1.5. Структура FDD методології

Переваги FDD:

- Фокус на невеликих, вимірюваних функціях забезпечує високу видимість прогресу проекту.

- Методологія добре підходить для великих команд, оскільки дозволяє паралельно працювати над різними функціями.

- Процес є передбачуваним та повторюваним, що спрощує управління проектом.

- FDD є гнучкою, що дозволяє швидко адаптуватися до змін.

Недоліки FDD:

- Методологія може ігнорувати нефункціональні вимоги, такі як продуктивність або безпека, які не є прямими "функціями".

- Успішне застосування FDD вимагає наявності досвідчених розробників, які можуть швидко та якісно створювати код.

FDD — це ефективна методологія для проектів, де пріоритетом є швидке та інкрементальне створення працюючого продукту.

Dynamic System Development Method (DSDM) — фреймова методологія для швидкої розробки додатків (RAD). Вона фокусується на роботі в межах фіксованих ресурсів та часу, а обсяг функціональності є змінною. DSDM включає п'ять фаз, три з яких є ітеративними: дослідження можливості, бізнес-дослідження, ітерація функціональної моделі, ітерація проектування та побудови, імплементація.

Існують також інші підходи, такі як Agile-моделювання (гнучкий підхід до виконання моделювання) та Прагматичне програмування (набір найкращих практик програмування). У наступних розділах буде надано детальніший опис методологій, що використовуються в конкретних проектах, оскільки для подальшого викладу необхідне їх глибоке розуміння.

1.5. Концепція та методологія Extreme Programming

Extreme Programming (XP) є гнучкою методологією, що базується на наборі передових практик розробки програмного забезпечення. Її основою стали експериментальні проекти, зокрема Chrysler C3. XP формалізована через чотири ключові цінності та дванадцять основних практик, що визначають її структуру та підходи. Кожному члену команди XP призначаються певні ролі, що забезпечують ефективну взаємодію в проекті.

1.5.1. Ролі в Extreme Programming

У рамках XP-проекту кожен учасник може виконувати одну або кілька з наступних ролей:

- Відстежувач (Tracker) - контролює прогрес розробки, ініціює дії у разі відхилень від плану. Функціонально ця роль схожа на спрощену версію Scrum Master.

- Клієнт (Customer) - визначає вимоги, формулюючи історії користувачів, та створює функціональні тести. Клієнт також приймає рішення щодо пріоритетності функцій під час планування ітерацій.

- Програміст (Programmer) - відповідає за декомпозицію історій користувачів на завдання, їх оцінювання, написання модульних тестів і безпосередню реалізацію функціональності.

- Тренер (Coach) - спостерігає за дотриманням принципів XP.

- Тестувальник (Tester) - відповідає за виконання функціональних тестів, аналіз результатів та ініціювання коригуючих дій.

- Пророк (Prophet) - неофіційна, але часто присутня роль критика, що вказує на потенційні ризики та проблеми в проекті.

- Менеджер (Manager) - організовує зустрічі, веде протоколи та виконує інші адміністративні завдання, не втручаючись у процес розробки.

Ці ролі можуть бути гнучко розподілені між членами команди, і не обов'язково кожну роль виконує окрема людина.

1.5.2 Цінності та практики Extreme Programming

Чотири основні цінності XP — це комунікація, зворотний зв'язок, простота та мужність. Ці принципи втілені у дванадцятьох ключових практиках XP.

1. Гра планування (Planning Game): Дворівневий процес, що включає планування релізу та ітерацій. Клієнт пріоритизує історії користувачів, а програмісти оцінюють їх розмір.

2. Малі релізи (Small Releases): Часті випуски продукту (наприклад, кожні 2 місяці), що забезпечує швидкий зворотний зв'язок і полегшує управління проектом.

3. Метафора (Metaphor): Використання спільного поняття, що описує функціональність проекту, для забезпечення єдиного розуміння між клієнтом та командою.

4. Простий дизайн (Simple Design): Фокусування на найпростішому рішенні, яке відповідає поточним вимогам, уникаючи зайвої складності та надмірної функціональності.

5. Тестування (Testing): Застосування розробки через тестування (TDD), де модульні тести створюються до написання виробничого коду. Це забезпечує високу якість і мінімізує помилки.

6. Рефакторинг (Refactoring): Постійне поліпшення внутрішньої структури коду без зміни його зовнішньої поведінки, що підтримує його зрозумілість та гнучкість.

7. Парне програмування (Pair Programming): Дві особи працюють за одним комп'ютером (водій та навігатор). Дослідження [WU01] показують, що ця практика підвищує якість коду та ефективність, сприяючи поширенню знань.

8. Колективне володіння кодом (Collective Code Ownership): Дозвіл будь-якому члену команди вносити зміни в будь-яку частину коду, що вимагає ефективної системи контролю версій.

9. Безперервна інтеграція (Continuous Integration): Регулярне додавання нового коду до спільної кодової бази, що забезпечує часті збірки та раннє виявлення конфліктів.

10. Без надгодин (Sustainable Pace): Встановлення 40-годинного робочого тижня для запобігання вигоранню та зниженню продуктивності.

11. Клієнт на місці (On-Site Customer): Ідеальний, хоча і рідкісний, сценарій, коли представник замовника працює безпосередньо з командою для швидкого вирішення питань.

12. Стандарти кодування (Coding Standards): Використання єдиних правил і стандартів для підтримки читабельності та консистентності коду.

1.5.3 Життєвий цикл Extreme Programming

Життєвий цикл XP складається з п'яти фаз, візуалізованих на рис. 1.6.

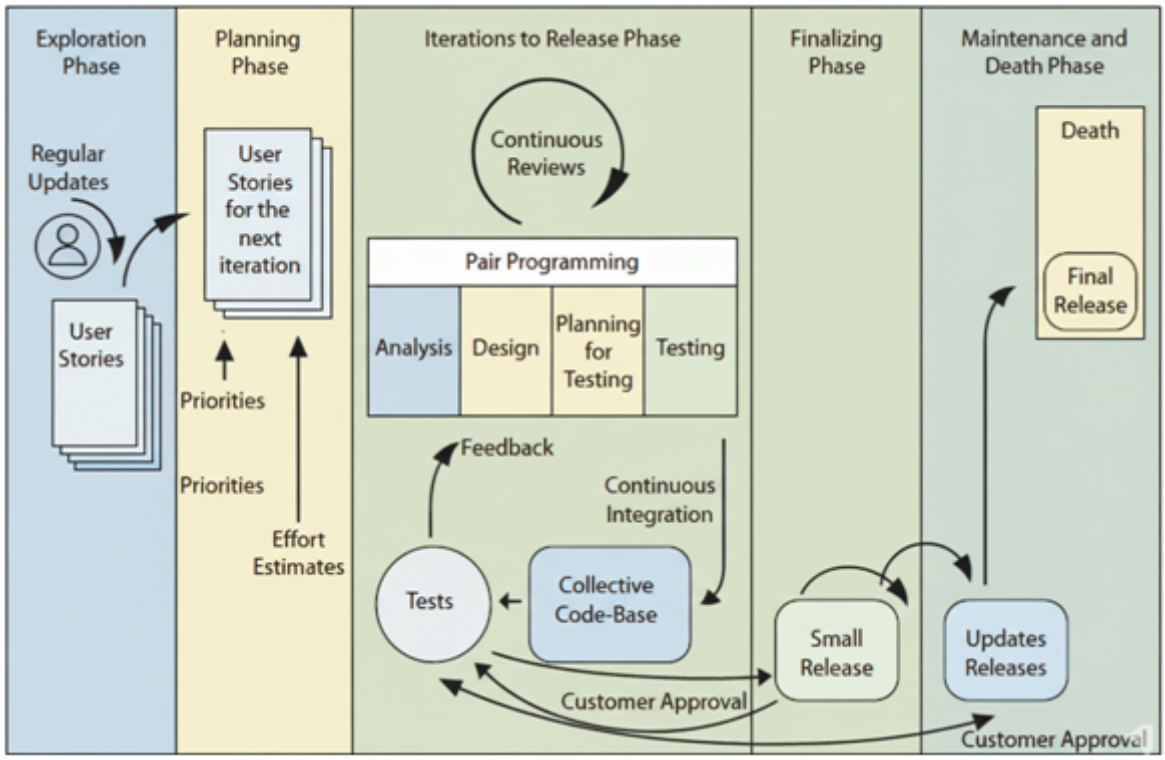

Рис. 1.6. Фази життєвого циклу XP

Фаза дослідження (Exploration) - клієнт формулює вимоги (історії користувачів), а розробники досліджують технології та створюють прототипи.

Фаза планування (Planning) - оцінка історій користувачів та визначення обсягу першого релізу.

Фаза ітерації до випуску (Iterations to Release) - повторювані ітерації розробки до моменту готовності першого релізу.

Фаза завершення (Release) - підготовка продукту до випуску, усунення остаточних недоліків та документування.

Фаза обслуговування та смерті (Maintenance & Death) - підтримка випущеного продукту, подальша розробка нових релізів. Фаза смерті настає, коли проект завершується або скасовується.

Висновки до розділу

У результаті огляду предметної області було визначено ключові принципи та історичні передумови становлення гнучких методологій управління програмними проєктами. Проведений аналіз дозволив систематизувати термінологію, порівняти особливості Scrum, XP та інших Agile-методологій, а також визначити їхні сильні й слабкі сторони. Це створило підґрунтя для подальшого вибору оптимальних методів планування та відстеження програмних проєктів.

РОЗДІЛ 2. МЕТОДОЛОГІЇ ТА МЕТОДИ ПЛАНУВАННЯ ПРОГРАМНИХ ПРОЕКТІВ

2.1. Теоретичні засади та інструменти контролю методології Scrum

Оскільки Scrum є однією з ключових методологій, що підтримуються багатьма системами планування програмного проекту, він потребує поглибленого розгляду.

2.1.1. Емпіричний підхід до розробки програмного забезпечення

Як було зазначено раніше, Scrum виник у відповідь на відсутність очікуваної продуктивності в багатьох об'єктно-орієнтованих проектах. Дослідження, проведене незалежними розробниками, зокрема Advanced Development Methods (ADM) та VMARK Software, базувалося на гіпотезі, що написання програмного забезпечення є творчим процесом, керованим правилами, а не жорстко визначеною послідовністю дій. Відомо, що розробка програмного забезпечення містить багато невизначених і неповторюваних процесів, а не добре визначених, повторюваних і передбачуваних, як у традиційній інженерії [8].

Цей висновок призвів до ідеї, що розробка програмного забезпечення має більше спільного з емпіричними процесами, ніж із визначеними.

Визначений процес є передбачуваним, може бути повністю автоматизованим і дає однаковий результат при кожному повторенні.

Емпіричний процес є "чорною скринькою", де відомі лише вхідні та вихідні дані, а внутрішні механізми непередбачувані. Управління такими процесами здійснюється через жорсткий контроль параметрів.

На основі цього дослідження було сформульовано низку вимог, яким має відповідати процес розробки:

- Толерантність до хаосу та здатність гнучко реагувати на непередбачувані події.

- Постійний моніторинг і корекція через механізми контролю.

- Визнання того, що продукт постійно еволюціонує.

- Максимізація комунікації та обміну інформацією.

- Створення оптимального програмного забезпечення з урахуванням обмежень.

2.1.2. Засоби контролю в Scrum

Scrum використовує ряд інструментів контролю для забезпечення відповідності вищезазначеним вимогам:

- Беклог (Backlog): Динамічний список всіх вимог до продукту. Його елементами є історії користувачів.

- Об'єкти та компоненти (Objects and Components): Самодостатні, повторно використовувані елементи системи.

- Пакети (Packages): Група об'єктів, що має високу внутрішню зв'язність і низьку зв'язність із зовнішніми елементами.

- Проблеми (Problems): Завдання, які команда повинна вирішити для реалізації історії користувача, включаючи дефекти (баги).

- Питання (Questions): Проблеми, які потребують вирішення перед початком реалізації.

- Рішення (Solutions) та Зміни (Changes): Реакція на проблеми та питання.

- Ризики (Risks): Потенційні загрози, пов'язані з історіями, проблемами або питаннями.

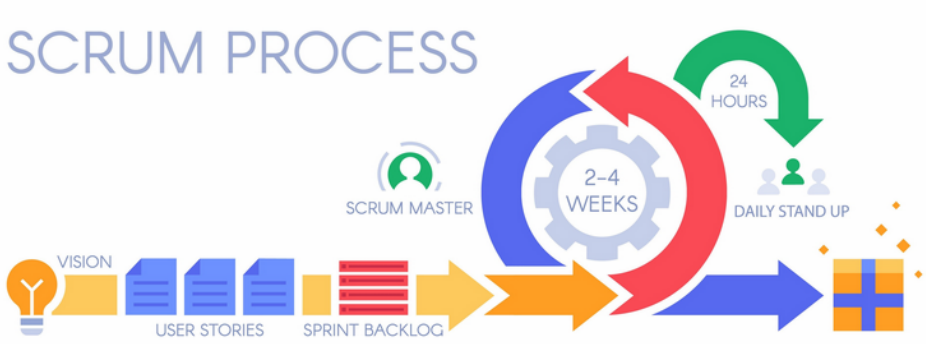

Рис. 2.1. Процес Scrum в загальному випадку

Найбільш важливими з цих інструментів є беклог та пов'язані з ним ризики.

2.1.3. Фази розробки Scrum

Процес Scrum поділяється на три основні фази:

- Фаза пре-гри (Pre-Game): Фаза планування, що включає дві підфази:

- Планування (Planning) - створення та оцінювання історій користувачів, формування команди та ресурсне забезпечення.

- Архітектура (Architecture) - високорівневе проєктування системи на основі поточного беклогу.

- Фаза розробки (Development): Найтриваліша фаза, що функціонує як "чорна скринька" емпіричного процесу. Розробка відбувається у спринтах — ітераціях фіксованої тривалості. В кінці кожного спринту продукт приростає новою функціональністю, яка переглядається на огляді спринту. Беклог продукту та системна архітектура постійно розвиваються, а власник продукту може змінювати беклог. Фаза триває, доки замовник не буде задоволений продуктом.

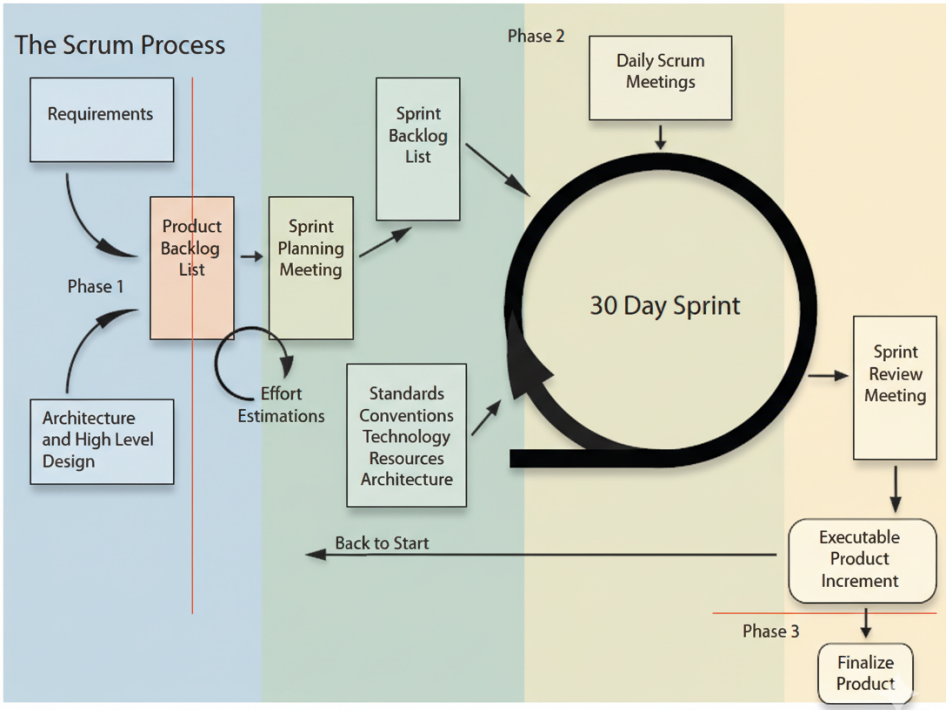

Рис. 2.2. Фази розробки Scrum

- Фаза пост-гри (Post-Game): Фаза впровадження, де продукт готується до випуску. Виконуються остаточні інтеграції та створюється документація. Продукт випускається після завершення всіх підготовчих робіт.

Візуальне представлення процесу Scrum наведено на рисунку 2.2.

2.2. Ролі та практики в методології Scrum

На відміну від Extreme Programming (XP), яка зосереджена на інженерних практиках, методологія Scrum акцентує увагу на управлінні та контролі процесу розробки. Це відображається у чітко визначених ролях та практиках.

2.2.1. Ролі та зацікавлені сторони

Scrum визначає три основні ролі.

Scrum-майстер (Scrum Master). Відповідає за впровадження методології Scrum. Його завдання полягає у фасилітації роботи команди, консультуванні всіх учасників проекту та усуненні перешкод, що перешкоджають продуктивності.

Власник продукту (Product Owner). Несе відповідальність за проект і його успіх. Він керує беклоґом продукту та визначає його пріоритети, діючи як зв'язок між зацікавленими сторонами та командою.

Scrum-команда (Scrum Team). Самоорганізована команда, що відповідає за досягнення цілей спринту. Члени команди беруть участь у плануванні, оцінюванні та розробці функціональності.

До ключових зацікавлених сторін належать:

- користувачі (Users) - особи, які будуть використовувати розроблену систему. Їхня участь у співпраці з замовником є важливою для забезпечення відповідності функціоналу реальним потребам.

- замовник (Customer) - фізична або юридична особа, що замовляє продукт і бере участь у формуванні беклогу.

- менеджмент (Management) - організаційна структура, яка приймає рішення щодо проекту, встановлює стандарти, контролює прогрес та здійснює інші управлінські функції.

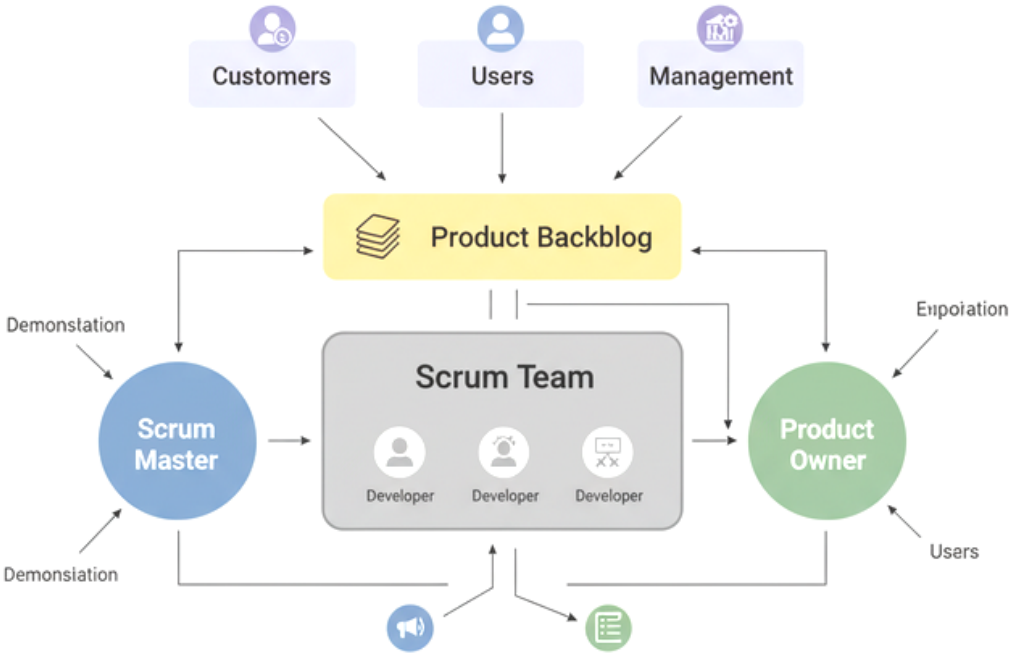

Рис. 2.3. Ролі в Scrum та їх взаємодія

2.2.2. Практики Scrum

Практики Scrum є інструментами управління, які забезпечують контроль над проектом. До основних практик належать:

- продуктовий беклог (Product Backlog) - список усіх відомих на поточний момент вимог до продукту, що складається з історій користувачів.

- оцінка зусиль (Effort Estimation) - процес оцінювання обсягу роботи, необхідної для реалізації кожної історії користувача. Оцінка також проводиться на рівні завдань на початку кожного спринту.

- спринт (Sprint) - ітерація в Scrum, що має фіксовану тривалість. Управління спринтом здійснюється за допомогою:

- зустрічі з планування спринту (Sprint Planning Meeting) - зустріч, що складається з двох фаз: визначення мети та обсягу спринту, а також детального планування інкременту продукту командою розробників.

- спринт-беклог (Sprint Backlog) - набір історій користувачів, які мають бути завершені в рамках поточного спринту.

- щоденної Scrum-зустрічі (Daily Scrum Meeting) - коротка щоденна зустріч, де кожен член команди звітує про свій прогрес, проблеми та плани, що сприяє швидкому вирішенню перешкод.

- зустрічі з огляду спринту (Sprint Review Meeting) - зустріч, на якій демонструються результати спринту, приймаються рішення щодо подальших кроків та оновлюється продуктовий беклог.

Ці практики є основою Scrum, забезпечуючи управління хаотичними процесами розробки. Комбінація управлінських практик Scrum з інженерними практиками XP відома як XP@Scrum.

Розвиток Scrum триває, і нові підходи, що стосуються підвищення ефективності, постійно з'являються.

2.3. Інтеграція Scrum та Extreme Programming: XP@Scrum

У попередніх розділах було розглянуто, що Extreme Programming (XP) акцентує увагу на інженерних практиках, що виникли з патернів кодування, тоді як Scrum фокусується на управлінні циклом розробки, що має коріння в японських процесах продуктової розробки. Ці методології ефективно доповнюють одна одну: практики XP компенсують нестачу інструментів для роботи з низькорівневими проблемами в Scrum, тоді як управлінські практики Scrum вирішують високо-рівневі питання планування в XP. Єдине їхнє перекриття, гра планування, було запозичене XP у Scrum для базового оцінювання та відстеження. Це перекриття не створює проблем, оскільки його реалізація ідентична в обох методологіях. Усі інші практики є унікальними для кожної методології. Взаємозв’язок практик XP та Scrum візуально представлений на рисунку 2.3, де червоне коло позначає XP, синє — Scrum, а фіолетове — зону перекриття.

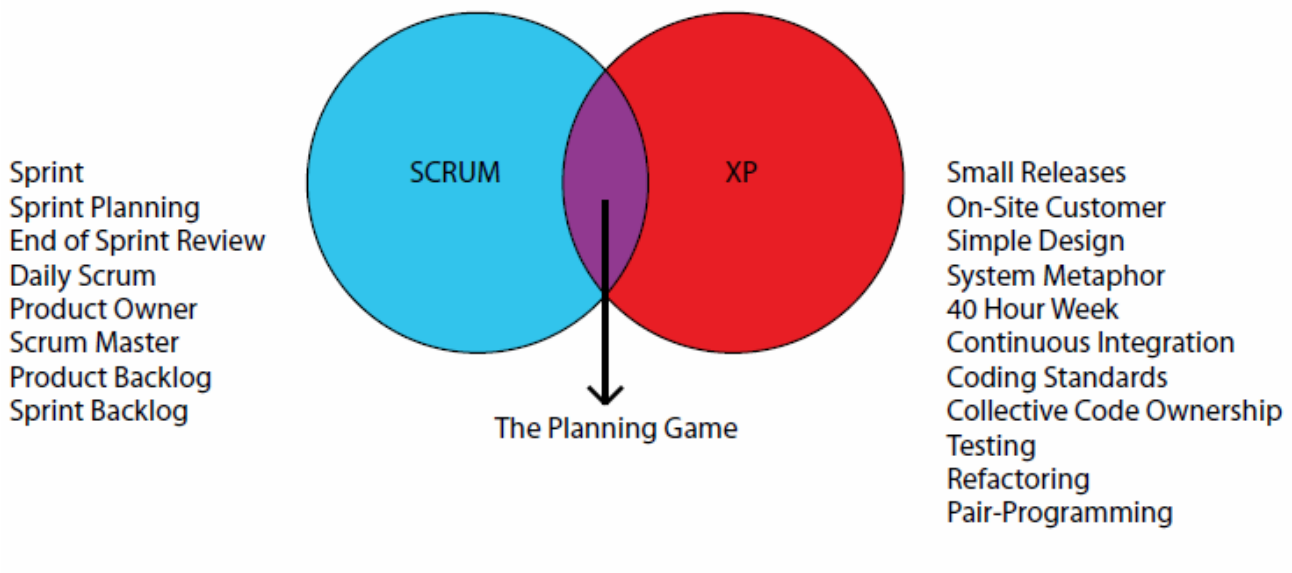

Рис. 2.3. Взаємозв'язок практик XP та Scrum

Часто компанії з розробки програмного забезпечення компанії використовують підхід XP@Scrum. Як і в більшості реальних імплементацій, вони не дотримуються XP в усіх деталях, а використовують практики, що найкраще відповідають потребам організації. Проте планування та відстеження повністю базується на Scrum, хоча і з додатковими процедурами, необхідними для підтримки сертифікації Capability Maturity Model Level 2.

2.3.1. Графічна візуалізація практик Extreme Programming (XP)

Для розуміння взаємодії XP та Scrum, практики XP можна згрупувати в чотири концентричні кола (рисунок 2.4), що представляють різні домени застосування.

Коло кодування (Coding Circle) - внутрішнє коло, що включає такі практики, як тестування, рефакторинг і парне програмування. Ці практики безпосередньо стосуються процесу написання коду.

Коло команди (Team Circle) - Наступне коло, що охоплює безперервну інтеграцію, стандарти кодування та колективне володіння кодом. Вони стосуються взаємодії та організації роботи команди.

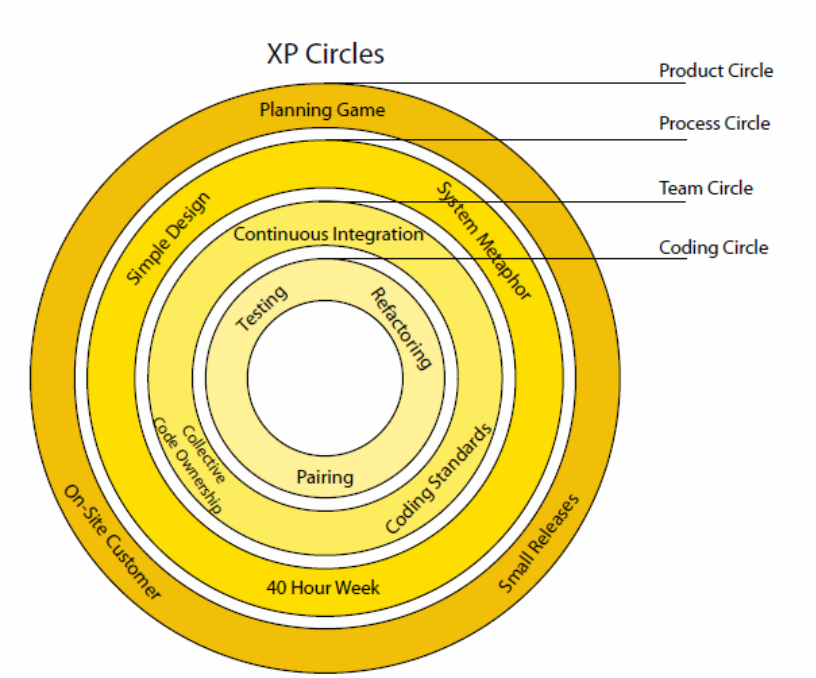

Рис. 2.4. Практики XP у вигляді концентричних кіл

Коло процесу (Process Circle) включає такі практики, як 40-годинний робочий тиждень, простий дизайн та метафора системи. Ці практики формують загальний підхід до розробки.

Коло продукту (Product Circle) - зовнішнє коло, що містить гру планування, клієнтів на місці та малі інкрементні релізи. Ці практики пов'язані з взаємодією з клієнтом та поставкою продукту.

2.3.2. Візуалізація інтеграції XP та Scrum

Оскільки Scrum і XP майже не перетинаються, їхня взаємодія може бути візуалізована як "обгортання" практик XP у фреймворк Scrum. На рисунку 2.5 показано, як практики XP інтегруються в життєвий цикл Scrum.

1. Щоденна робота.

Практики XP, що стосуються щоденної роботи (кола кодування та команди), обгорнуті в Щоденний Scrum (Daily Scrum). Це представлено у вигляді менших кіл у верхній частині рисунка.

2. Спринт.

Практики XP, що стосуються ітерації в цілому (кола процесу та продукту), обгорнуті в Спринт Scrum. Це показано у вигляді більших кіл у нижній частині рисунка.

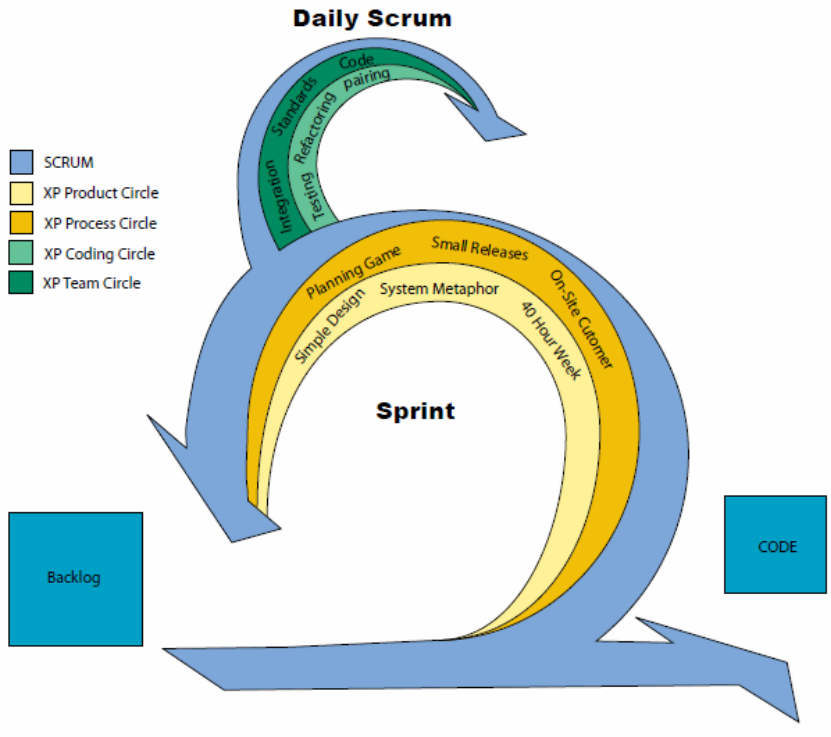

Рис. 2.5. Практики XP, об'єднані плануванням та відстеженням Scrum

Таким чином, XP@Scrum об'єднує інженерні практики XP (внутрішні кола) з підходами Scrum до планування та відстеження (зовнішні кола), створюючи цілісну та ефективну гібридну методологію.

2.4. Формування набору тестових даних

Для проведення порівняльного аналізу інструментів управління гнучкими проектами необхідний релевантний набір даних. Оскільки історії користувачів та завдання є основними одиницями планування та відстеження в методологіях Scrum та Extreme Programming (XP), ці елементи становлять ядро тестового набору. Оцінки трудомісткості та облік часу в цих даних представлені в годинах, що є уніфікованою мірою.

З метою забезпечення максимальної достовірності порівняння, тестові дані були сформовані на основі реального проекту, в рамках якого був розроблений інструмент планування проектів. Початково для аналізу використовуються дані з двох ітерацій, з можливістю залучення додаткових даних (ще двох ітерацій) у разі потреби. Для забезпечення чистоти даних та їх відповідності "ідеальному" сценарію, з набору були вилучені історії користувачів, які не стосувалися безпосередньої функціональності продукту, а використовувалися для обліку накладних витрат (наприклад, документація стажування).

2.4.1. Ієрархічна структура робочих елементів

Структура робочих елементів у гнучкому проекті, що використовує методологію XP@Scrum, відповідає загальноприйнятій структурі розбиття робіт (WBS). Ця ієрархія універсальна, оскільки відображає реальні процеси розробки програмного забезпечення.

Проект (Project) - контейнер верхнього рівня, що містить усі робочі елементи, пов'язані з розробкою системи. Як правило, проект складається з декількох релізів.

Реліз (Release) - функціонально завершена частина програмного забезпечення. Релізи є найвищим рівнем для планування та відстеження. Вони складаються з кількох ітерацій, а їх тривалість у гнучких проектах становить, як правило, 3–6 місяців. Функціональність релізу визначається сукупністю історій користувачів, що формують беклог релізу (Release Backlog).

Ітерація (Iteration) - інтервал часу (від 1 до 4 тижнів), протягом якого реалізується частина історій з беклогу релізу. Ітерації є основним рівнем для детального планування. Гнучкість дозволяє замовнику додавати або видаляти історії з беклогу релізу, але зміна історій у поточній ітерації не майже допускається.

Історія користувача (User Story) - одиниця функціональності, що оцінюється в абстрактних мірах — ідеальних годинах або балах історії. Оцінка відображає відносний розмір і складність роботи.

Завдання (Task) - декомпозиція історії користувача. Кожне завдання має власну оцінку в ідеальних годинах. Завдання також включають дані про витрачений час та кількість годин до завершення (Togo).

Користувачі (Users) - Особи, що виконують різні ролі в проекті. Вони можуть бути об'єднані в команди, а їхні ролі визначені відповідно до методології, як описано на початку розділу.

2.4.2. Припущення дослідження

Для цілей тестування були прийняті такі припущення:

- Над проектом працюють два розробники з однаковими навичками та кваліфікацією, кожен з яких присвячує проекту 40 годин на тиждень.
- Тривалість ітерації становить два тижні.
- Тестовий період початково охоплює дві ітерації, з можливістю його розширення.
- У проекті бере участь один представник замовника, який завжди доступний.

2.5. Аналіз тестових даних та критерії оцінки

2.5.1. Структура тестових даних

Набір даних охоплює чотири ітерації, кожна з яких містить приблизно десять історій користувачів та двадцять пов'язаних завдань. Для цілей тестування буде створено один проект та один реліз, оскільки реліз є найвищим рівнем планування, що підтримується всіма аналізованими інструментами.

Таблиці, що містять приклади даних (таблиці 2.1 та 2.2), ілюструють ключові параметри, такі як:

- Опис історії користувача: Деталізація функціональності.

- Оцінка (Estimation): Початкова оцінка трудомісткості історії.

- Загальні початкові зусилля (Total Initial Effort): Сумарна оцінка завдань, що входять до історії.

- Загальні використані (Total Consumed): Фактичні витрати часу.

- Загальні останні ToGo (Total Last ToGo): Залишок часу до завершення.

- Прийняття замовником (Customer Acceptance): Підтвердження клієнтом завершеної функціональності.

Таблиця 2.1.

Список беклогу спринту

Структура розбиття робіт тиждень 21/22	**(годин)**	**(годин)**	**(годин)**	**(годин)**	
Опис історії користувача	Оцінка	Загальні початкові зусилля	Загальні використані	Загальні останні ToGo	CA*
1. Завантажити фотографії користувачів	2	2	2	0	7
2. Графічний огляд відсутності користувачів з можливістю друку через PDF-файл (2)	48	16	16	0	0
3. Забезпечення якості тиждень 21/22			0	0	7
4. Управління конфігурацією тиждень 21/22			0	0	7
5. Непрямі години тиждень 21/22			3.5	0	7

Таблиця 2.2.

Список задач

2. Графічний огляд відсутності користувачів з можливістю друку через PDF-файл (2)				
Створити графік		8	14	0
Включити таблицю балансу відпусток у PDF-файл (праворуч від легенди)		6	1.6	0
Видалити другий фільтр для балансу відпусток (використовувати перший фільтр замість цього)		2	0.6	0

2.5.2. Особливості оцінювання

Важливо відзначити, що оцінки, отримані на рівні завдань, вважаються більш точними, ніж оцінки на рівні історій користувачів, оскільки завдання є меншими та менш абстрактними. У випадку розбіжностей, для аналізу використовується оцінка, отримана шляхом сумування оцінок завдань.

У випадках, коли інструменти є хостинговими рішеннями з обмеженим доступом, для їх оцінки будуть використовуватися надані тестові проекти. Ці проекти, засновані на реальних даних, є достатньо репрезентативними для порівняння. Будь-який вплив використання альтернативних даних на результати аналізу буде зафіксовано в розділі результатів.

2.5.3. Критерії оцінки

Для всебічного аналізу визначено такі критерії оцінки.

1) Базова функціональність: Оцінка наявності та реалізації ключових елементів управління проектом, таких як:

- Проекти, Релізи, Ітерації
- Історії користувачів, Завдання
- Управління користувачами

2) Абстрактні аспекти: Аналіз реалізації таких критичних аспектів гнучких методологій:

- Планування: На рівні релізів та ітерацій.
- Реалізація: Як інструмент підтримує виконання завдань.
- Метрики: Надання даних для аналізу.
- Відстеження: Можливості моніторингу прогресу на основі релізів та ітерацій.

3) Зручність використання (Usability): Оцінка загальної ефективності та комфорту роботи з інструментом, що включає:

- Процес встановлення
- Наявність довідкової документації
- Швидкість відгуку

- Зручність інтерфейсу (макет/UI)

Для компанії ключовими критеріями є зручність використання, а також ефективність реалізації функцій планування та відстеження. Аналіз буде сфокусований на базових завданнях, спільних для всіх гнучких проектів.

Кожен інструмент буде порівняно з пропонованим для визначення його переваг та недоліків. Зокрема, буде встановлено, чи пропонує інструмент унікальні, корисні функції; чи реалізує існуючі функції краще, ніж пропонований; та чи має він недоліки, яких слід уникати. Також буде досліджено, наскільки кожен інструмент дотримується чистої методології Agile.

Висновки до розділу

Дослідження методів і методологій планування показало, що Scrum і Extreme Programming ефективно доповнюють одне одного, формуючи інтегрований підхід до управління програмними проєктами. Сформовано критерії оцінювання ефективності таких систем, визначено структуру та особливості тестових даних, що забезпечило можливість верифікації результатів. Отримані положення стали основою для розробки системної моделі управління.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ ТА МЕТОДОЛОГІЙ В СИСТЕМІ ПЛАНУВАННЯ ТА ВІДСТЕЖЕННЯ ПРОГРАМНИХ ПРОЕКТІВ

3.1. Загальна характеристика та функціональні особливості системи планування проектів

Система планування і трекінгу програмних проектів (СПТПП) — це інструмент, розроблений для планування та відстеження проектів розробки програмного забезпечення. Його функціональність поєднує підтримку методологій Agile, вимоги сертифікації Capability Maturity Model (CMM) рівня 2 та особливості, специфічні для організації Philips. Ця комбінація вносить зміни до базового Agile-фреймворку, які будуть розглянуті в подальшому аналізі.

3.1.1. Методологія аналізу

Для оцінки системи буде використано дані з поточного дослідницького проекту, оскільки інструмент виконує розрахунки на стороні сервера, що ускладнює симуляцію. Структура цього проекту (невеликий проект з одним дослідником) подібна до тестового випадку, що робить його придатним для аналізу. В огляді системи буде приділена увага субоптимальним реалізаціям функцій, які вказують на проблемні місця, а також добре реалізованим функціям, що можуть слугувати основою для рекомендацій щодо змін.

Хоча система планування і трекінгу програмних проектів є інструментом для планування проектів, але в ньому наявні численні функції, специфічні для внутрішнього управління, такі як бюджетування, бронювання кімнат та управління компетенціями. Ці додаткові функції, які не стосуються безпосередньо планування та відстеження, не будуть оцінюватися, за винятком тих, що мають прямий вплив на ці процеси.

Процес інсталяції системи є відносно простим для користувачів, які мають доступ до веб-сервера та бази даних MySQL. Однак ініціалізація бази даних може бути складною для непідготовлених користувачів і потребує автоматизації.

3.1.2. Додаткові функції

Система включає ряд додаткових функцій, які, хоч і не є основними для Agile-планування, можуть бути корисними в управлінні проектами. До них належать:

- Модульні тести - функція, що дозволяє перевіряти цілісність коду після змін.

- Контрольний список забезпечення якості (QA) - функція, що є вимогою для сертифікації CMM рівня 2.

- Функції для управління внутрішніми ресурсами.

- Огляд компетенцій та вартості курсів - інструменти для управління персоналом.

- Управління ризиками та бюджетом - функції для моніторингу проектних показників.

Особливо корисними є такі функції:

1. Відсутності.

Система дозволяє користувачам зазначати плановану відсутність. Це коригує доступний час для виконання завдань в ітерації, що призводить до точнішого планування та відстеження.

2. Навантаження (Load).

Сторінка, що візуалізує навантаження на членів команди, порівнюючи призначену роботу з їх історичною продуктивністю. Це допомагає виявити як перевантаження, так і недостатню завантаженість.

3. Розподіл ресурсів (Resource Allocation).

Графік, що відображає завантаження співробітників, допомагаючи керівникам проектів ефективно планувати нові проекти.

4. Компетенції (Competences).

Функція, що відображає навички співробітників, що в поєднанні з даними про доступність ресурсів є потужним інструментом для формування команди.

Більшість цих додаткових функцій зосереджені на початковому плануванні та управлінні ресурсами. Хоча вони є цінними, їхня інтеграція в користувацький інтерфейс системи наразі не є оптимальною.

3.2. Представлення функцій планування в системі

3.2.1. Планування проекту

Цей розділ присвячений детальному аналізу функцій планування в системі, з фокусом на послідовності дій: від створення проекту до формування ітерацій. Особлива увага приділяється юзабіліті та зворотному зв'язку, що надається користувачеві.

Створення проекту та призначення ресурсів є початковим етапом. Адміністратор може ініціювати новий проект, заповнивши детальні дані про клієнта, дати, контакти та опис. Однак процес призначення користувачів є неоптимальним: він вимагає навігації між трьома окремими сторінками (список користувачів, розподіл ресурсів та сторінка призначень). Це ускладнює процес, хоча рідкість цієї операції зменшує її вплив на загальне враження. Оптимальною альтернативою було б інтеграція цього функціоналу на сторінку проекту, де можна було б візуально, наприклад, через drag-and-drop, призначати доступних співробітників.

Інформація про проекти доступна на трьох сторінках:

- Основна сторінка проекту: Відображає базові дані обраного проекту та дозволяє редагування.

- Сторінка огляду: Надає зведений список усіх проектів (поточних, запланованих, минулих).

- Сторінка звітів: Призначена для подання заповнених шаблонів звітів про статус.

- Відсутність єдиної сторінки з поточним статусом для кожного проекту є недоліком, особливо враховуючи наявність необхідних даних у базі.

3.2.2. Беклог та ітерації

Беклог в PPTS (рисунок 3.1) служить сховищем для історій користувачів. Історії можна додавати та пріоритизувати, переміщуючи їх у списку, що відповідає практиці Scrum.

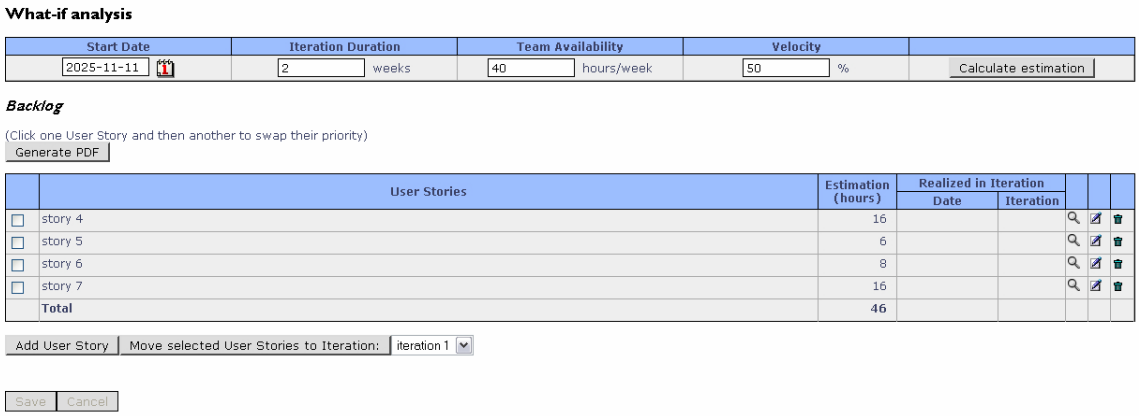

Рис. 3.1. Сторінка беклогу

Система не підтримує концепцію релізів. Це означає, що ітерації прив'язані безпосередньо до проекту. Така структура є проблемою для довгострокових проектів, що потребують багаторівневого планування, де релізи слугують важливими віхами. Додавання функціоналу для управління релізами значно підвищило б цінність інструменту для великих проектів.

Планування ітерацій є досить зручним. Користувачеві потрібно тільки перемикатися між сторінками для переміщення історій користувачів з беклогу в ітерацію. Візуальний зворотний зв'язок досить обмежений: на сторінці ітерації відображається лише оцінка в годинах, але не доступні

ресурси. Для отримання інформації про ресурси користувач повинен перейти на сторінку спадання (burn-down chart) або метрик.

3.2.3. Історії та завдання

Історії користувачів представляють функціональні вимоги, визначені замовником. Їх можна створювати на сторінках беклогу або структури розбиття робіт (WBS). Пріоритизація історій можлива лише в беклозі.

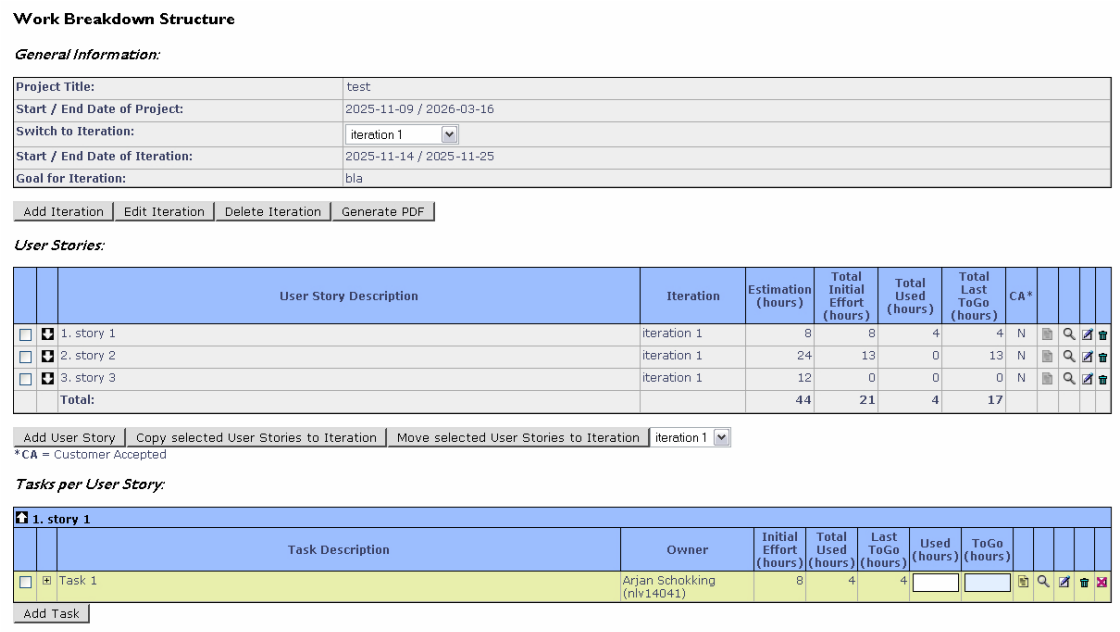

Рис. 3.2. Сторінка структури розподілу робіт

Завдання є декомпозицією історій на менші, більш керовані кроки. Кожне завдання має власну оцінку, яка, завдяки меншому обсягу роботи, як правило, є точнішою, ніж оцінка історії. Ці оцінки використовуються для точнішого розрахунку загальної трудомісткості.

3.3. Аналіз функцій відстеження в системі

3.3.1. Відстеження проектів

Відстеження є критично важливим компонентом гнучких методологій, оскільки воно забезпечує контроль над процесом розробки. В умовах, коли

кінцевий результат не визначений на початковому етапі, моніторинг прогресу дозволяє вчасно виявляти відхилення та вживати коригувальні заходи.

В системі відсутній механізм відстеження на рівні проекту чи релізу. Це є значним недоліком, оскільки релізи відіграють ключову роль у плануванні довгострокових проектів. Додавання функціоналу для відстеження релізів є пріоритетною рекомендацією для вдосконалення.

3.3.2. Відстеження на рівні ітерації

Основні функції відстеження в PPTS зосереджені на рівні ітерацій. Статус ітерації можна оцінити за допомогою кількох сторінок:

- WBS: Відображає чисельні дані, але їх інтерпретація є складною через формат.

- Звіт про статус: Надає огляд прогресу історій користувачів та завдань, без вказання часових рамок.

User Stories:

	User Story Description	Esti-mation	Initial Effort	Week 46 (14-Nov)		Week 47 (21-Nov)		Total Used	User Story Status
				Used	ToGo	Used	ToGo		
⊞	1. story 1	8	8	0	8	4	4	4	50%
⊞	2. story 2	24	13	0	13	0	13	0	0%
⊞	3. story 3	12	No Tasks defined yet for this User Story						
	Total	**44**	**21**	**0**	**21**	**4**	**17**	**4**	19%

Tasks per User Story:

⊟ 1. story 1

Task Description	Task Owner	Initial Effort	Week 46 (14-Nov)		Week 47 (21-Nov)		Total Used	Task Status
			Used	ToGo	Used	ToGo		
Task 1	AS	8	0	8	4	4	4	50%
Total		**8**	**0**	8	**4**	4	**4**	50%

Рис. 3.3. Звіт про статус

- Графік спадання (Burndown Chart): Найефективніший інструмент для візуалізації прогресу. Графік показує обсяг роботи, що залишився, дозволяючи оцінити ймовірність завершення ітерації вчасно та коригувати її обсяг. Цей графік базується на оцінках завдань і є цінним індикатором ефективності команди.

- Сторінка метрик: Надає додаткові чисельні дані, що дублюють інформацію з графіка спадання та можуть бути інтегровані в єдиний звіт.

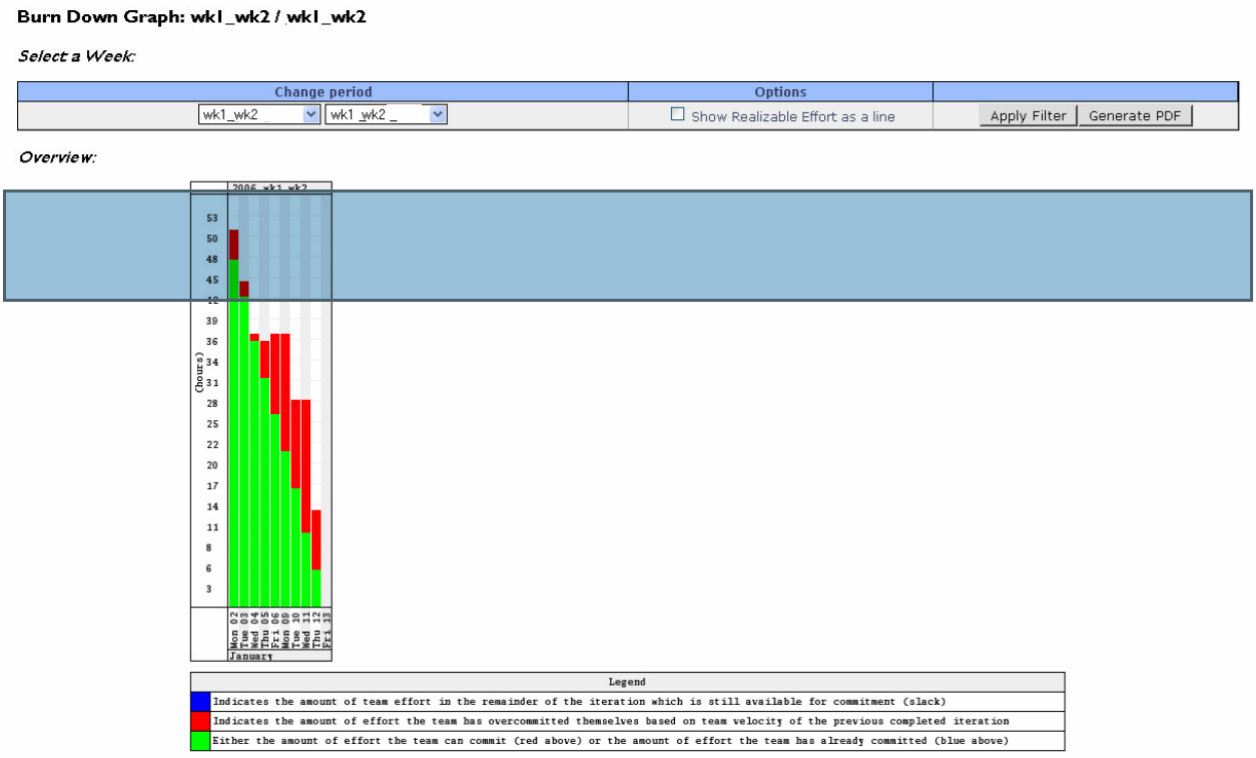

Рис. 3.4. Графік спадання (Burndown Chart)

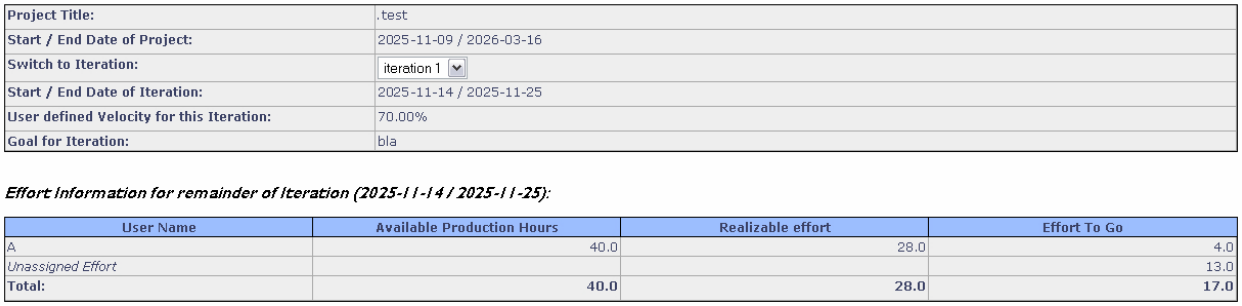

Рис. 3.5. Сторінка представлення метрик

3.3.3. Відстеження історій користувачів та завдань

Хоча історії користувачів є основною одиницею планування, завдання є найточнішим рівнем для відстеження. Прогрес завдань відображається в звіті про статус, однак без прямого зв'язку з часом, що залишився в ітерації. Основною одиницею відстеження є кількість годин, витрачених на завдання, та годин, що залишилися (ToGo).

Для забезпечення точного відстеження критично важливо, щоб розробники регулярно оновлювали свої витрати часу. Несвоєчасне внесення даних призводить до неточних показників проекту. Щоб вирішити цю проблему, можна впровадити систему сповіщень, яка б нагадувала користувачам про необхідність заповнення годин.

Незважаючи на наявність деяких ефективних інструментів, таких як графік спадання, система на даному етапі має значні недоліки у функціях відстеження, зокрема, через відсутність механізмів для моніторингу прогресу на рівнях проекту та релізу. Ці обмеження можуть бути усунені шляхом внесення змін у структуру та інтерфейс системи.

3.3.4. Інші аспекти функціональності

Цей розділ присвячений оцінці додаткових, але важливих аспектів інструменту, які не увійшли до попередніх категорій.

Прозорість інструменту для користувачів, обізнаних з методологіями Scrum та Extreme Programming (XP), є відносно високою. Однак для нових користувачів, які не мають такого досвіду, відсутність посібника користувача створює значні перешкоди для розуміння функціональності. Крім того, необхідність навігації між кількома сторінками для виконання типових завдань знижує прозорість та інтуїтивність роботи з системою.

Документація для системи практично відсутня, за винятком короткого посібника зі встановлення. Відсутність підказок (tooltips) на неочевидних елементах інтерфейсу вимагає від користувачів робити припущення, що є неприпустимим для професійного інструменту. Хоча графіки метрик мають чіткі позначення, загальна відсутність документації є серйозною проблемою, що обмежує використання інструменту поза межами вузького кола знайомих з ним розробників.

Реактивність системи в цілому задовільна, за винятком операцій, пов'язаних з генерацією великих графіків (наприклад, графіків розподілу

ресурсів). Це спричинено неефективною реалізацією, що генерує надмірну кількість запитів до бази даних SQL.

Дана система не має вбудованої системи відстеження помилок. Замість цього він інтегрується із зовнішніми базами даних, такими як Mantis або Bugzilla, дозволяючи імпортувати помилки як історії або завдання. Дана система є інструментом, що підтримує гібридну методологію XP@Scrum. Логічно, що інструмент фокусується більше на управлінських та організаційних аспектах Scrum, ніж на інженерних практиках XP. Це відповідає його призначенню як системи для планування та відстеження проектів.

Загальний аналіз системи виділяє кілька ключових сфер, що потребують вдосконалення. Наступні пункти є основою для порівняльного аналізу з іншими інструментами.

Процес планування проектів, особливо додавання користувачів, може бути спрощено. Відсутність підтримки релізів як окремого рівня планування є суттєвим недоліком. Планування ітерацій також потребує оптимізації через його громіздкість.

Необхідно впровадити відстеження на вищих рівнях абстракції — проекту та релізу. Крім того, візуальне представлення даних, що вже доступні в системі, може бути покращено для забезпечення більшої зрозумілості.

Поліпшення функцій планування та відстеження призведе до підвищення інтуїтивності та прозорості. Додавання рівня релізів також сприятиме кращому розумінню загальної структури великих проектів.

3.4. Методологічні основи та системна модель побудови фреймворку системи планування проектів

Дослідницький фреймворк є теоретичною та концептуальною структурою, яка скеровує наукове дослідження. Він включає в себе

сукупність припущень, концепцій, цінностей та практик, що визначають межі дослідження та надають йому методологічне керівництво. Цей фреймворк допомагає у формулюванні дослідницької проблеми, ідентифікації ключових змінних та їх взаємозв'язків, а також у виборі методів збору та аналізу даних [12]. Як невід'ємна частина дослідницького процесу, фреймворк забезпечує структуру та логічну послідовність для дослідження, дозволяючи дослідникам осмислити зібрані дані та зробити обґрунтовані висновки. Дослідницький фреймворк, використаний у цьому дослідженні, зображено на рисунку 3.6.

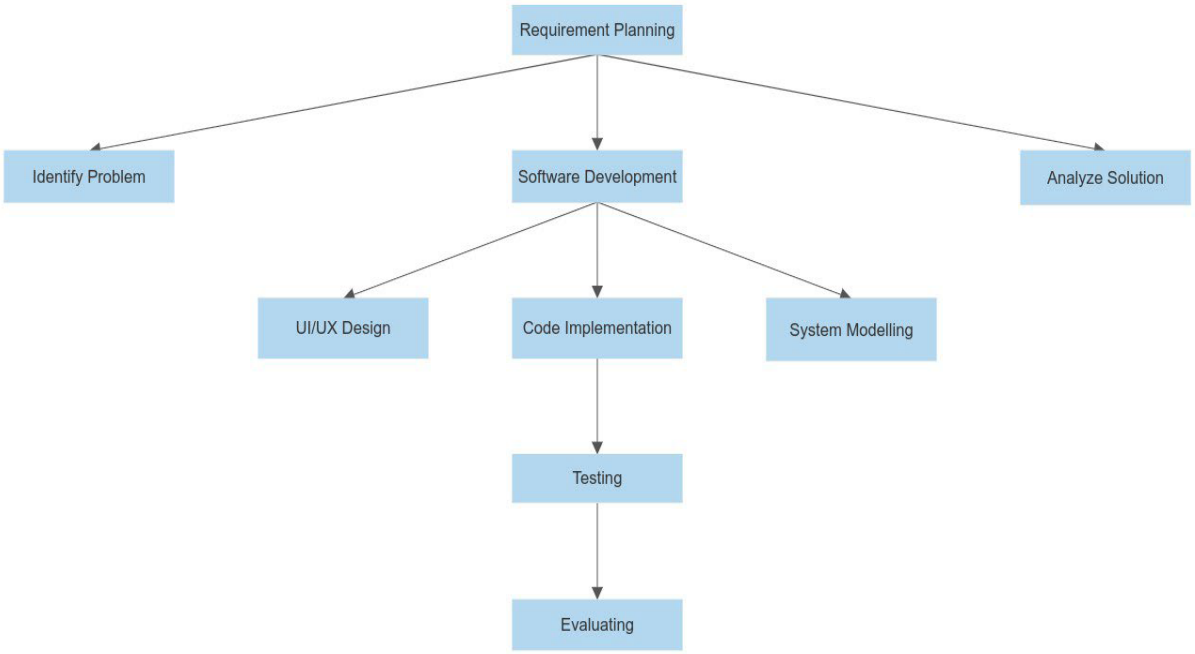

Рис. 3.6. Структура фреймворку

Розробка програмного забезпечення вимагає застосування специфічної методології та системної моделі. Методологія визначає підхід або послідовність кроків, яких дотримуються в процесі розробки. Системна модель, у свою чергу, описує структуру та функціональність системи.

Даний фреймворк включає наступні ключові етапи:

- UI/UX Дизайн: На цьому етапі розробляється зовнішній вигляд та робочий процес системи, що забезпечує зручність для користувача.

- Імплементація (Implementation): На цьому етапі відбувається написання та інтеграція програмного коду, що втілює розроблений дизайн.

- Тестування та оцінка (Testing and Evaluation): На завершальному етапі програмне забезпечення перевіряється на відповідність функціональним вимогам та оцінюється його ефективність у вирішенні поставленої проблеми.

Загалом, цей дослідницький фреймворк надає чітку та організовану структуру для наукової роботи, скеровуючи через усі необхідні етапи процесу розробки програмного забезпечення та гарантуючи, що всі кроки виконані для створення функціонального та ефективного продукту. Він охоплює всі необхідні етапи процесу розробки програмного забезпечення.

3.5. Методологія управління програмними проектами

Сучасний бізнес-світ демонструє, що традиційні підходи до управління проектами вже не є ефективними. Ці методології часто характеризуються тривалими термінами виконання та негнучкістю до змін, які зазвичай виникають на пізніх етапах, що ускладнює досягнення цілей замовника. Нові методології, на противагу, акцентують на швидкому досягненні результатів шляхом поставки відчутних продуктів. Ця відмінність у підходах суттєво впливає на вибір методології розробки програмного забезпечення.

Традиційна методологія водоспаду (Waterfall), яка фокусується на обсязі проекту для визначення вартості та термінів, є ефективною в передбачуваних умовах. Однак вона втрачає свою актуальність у сучасному, непередбачуваному світі. Тривале виконання критичних завдань у методології водоспаду часто призводить до затримок і накопичення незавершених робіт наприкінці проекту [21].

Методології, такі як PRINCE2 та PMBOK, розвинулися з водоспадної моделі і є одними з найпоширеніших у Європі та Північній Америці. Водночас, класичний водоспад з його чіткими фазами та детальною документацією все ще широко використовується, зокрема в офшорних

проектах розробки програмного забезпечення за контрактом. Усі проекти, які розглядалися в даному дослідженні, були реалізовані за методологією водоспаду для забезпечення однорідності факторів середовища розробки.

Традиційна методологія управління проектами, що підходить для малих, середніх і великих проектів, складається з послідовності етапів:

- Фаза ініціації (Initiation phase): На цьому етапі проводиться дослідження доцільності проекту. Проект вважається доцільним, якщо він відповідає критеріям, визначеним у трикутнику управління проектами:

- Приносить користь клієнтам.

- Вирішує проблеми, з якими стикається замовник.

- Може бути реалізований в межах очікуваних термінів, бюджету та наявних ресурсів.

- Фаза планування або дизайну (Planning or design phase): Основна відповідальність за цей етап лежить на команді впровадження проекту. На цьому етапі створюються деталізовані плани та дизайн системи.

- Фаза виконання або виробництва (Execution or production stage): Це фаза, де відбувається безпосередня реалізація завдань для створення програмного продукту згідно з вимогами. Вона включає:

- Програмування (Development)

- Тестування (Testing)

- Забезпечення якості (Quality assurance, QA)

- Документування (Documentation)

- Моніторинг та контроль (Monitoring and controlling systems): На цьому етапі здійснюється регулярний нагляд за виконанням проекту. Це дозволяє вчасно виявляти відхилення від плану та вживати коригувальних заходів. Моніторинг включає:

- Вимірювання прогресу

- Контроль змін

- Ідентифікацію коригувальних дій

- Керування змінами, що відхиляються від затвердженого плану.

- Фаза завершення (Completion stage): Ця фаза настає після передачі програмного продукту замовнику. Вона включає:

- Закриття проекту: Оцінка результатів, офіційна передача та припинення дії контракту.

- Перехід до етапу підтримки (maintenance): Продукт переходить у фазу обслуговування, яка може бути продовжена новим контрактом розробки.

3.6. Проектування системи управління проектами

Unified Modeling Language (UML) є де-факто стандартом у сфері аналізу та проектування програмного забезпечення [27]. Ця мова, надає графічний інструментарій для моделювання артефактів програмних систем на різних рівнях абстракції, включаючи вимоги, дизайн, реалізацію та тестування. UML широко визнана як стандарт для об'єктно-орієнтованого аналізу та проектування, а її діаграми слугують ключовим засобом комунікації між замовниками, розробниками та іншими зацікавленими сторонами.

Діаграми варіантів використання (Use Case Diagrams), як частина UML, є критичним елементом моделювання. Вони візуалізують взаємодію між акторами та варіантами використання, що дозволяє фіксувати високорівневі функціональні вимоги та цілі користувачів.

На рисунку 3.7 представлена діаграма варіантів використання, що моделює функціональність системи управління проектами. Дизайн поділено на два основні типи акторів: менеджер проекту (PM) та програміст. Пунктирна лінія, що з'єднує PM з варіантами використання, підкреслює його основну роль як ініціатора та контролера більшості системних операцій. На противагу, програміст має обмежений доступ, що відповідає його сфокусованим завданням.

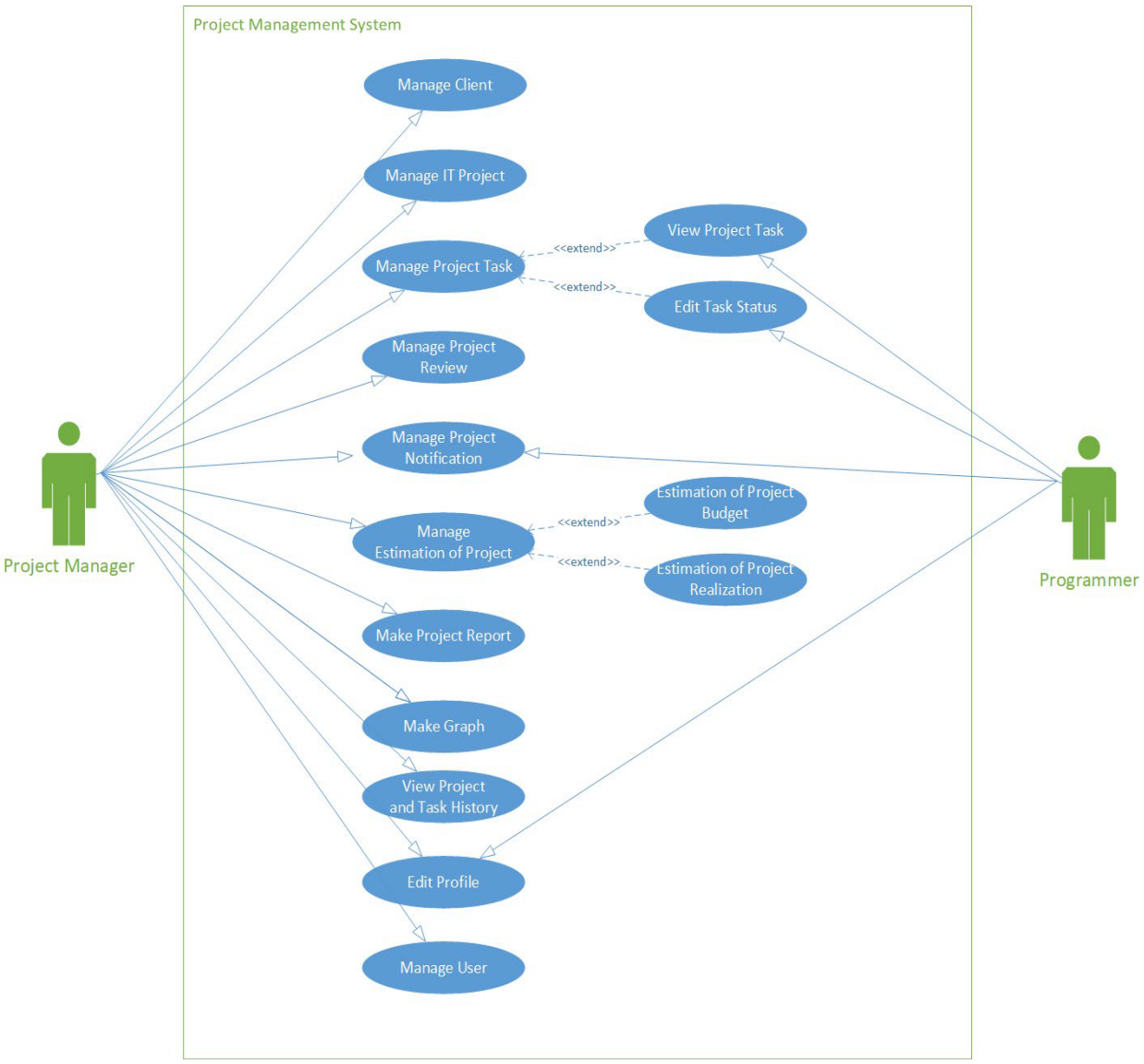

Рис. 3.7. Діаграма варіантів використання

Діаграма ілюструє, як ці два актори взаємодіють із системою:

- Управління клієнтами (Manage Client): Менеджер проєкту може переглядати та керувати інформацією про клієнтів.

- Управління IT-проєктами (Manage IT Project): Ця функція дозволяє менеджеру проєкту створювати, редагувати та видаляти проєкти.

- Управління завданнями проєкту (Manage Project Task): Менеджер проєкту може створювати та керувати завданнями, а також призначати їх програмістам. Цей варіант використання розширюється на Перегляд завдання проєкту (View Project Task) та Редагування статусу завдання (Edit Task Status), які можуть виконуватися обома акторами.

- Управління оглядами проєкту (Manage Project Review): Менеджер проєкту може працювати з оглядами та відгуками.

- Управління сповіщеннями проєкту (Manage Project Notification): Менеджер проєкту може керувати сповіщеннями та попередженнями.

- Управління оцінкою проєкту (Manage Estimation of Project): Менеджер проєкту має можливість керувати оцінками. Цей варіант використання розширюється на Оцінку бюджету проєкту (Estimation of Project Budget) та Оцінку реалізації проєкту (Estimation of Project Realization).

- Створення звіту проєкту (Make Project Report): Менеджер проєкту може створювати звіти для аналізу проєкту.

- Створення графіків (Make Graph): Ця функція дозволяє менеджеру проєкту візуалізувати дані проєкту.

- Перегляд історії проєкту та завдань (View Project and Task History): Менеджер проєкту може переглядати минулі дані.

- Редагування профілю (Edit Profile): Обидва актори можуть редагувати свої профілі.

- Керування користувачами (Manage User): Менеджер проєкту може керувати командою, включаючи програмістів та їхні завдання.

3.6.1. Імплементація та архітектура системи

Реалізація системи здійснювалася у формі веб-додатку. Для розробки програмного коду використовувалися PHP, HTML, CSS, JavaScript та фреймворки Bootstrap і CodeIgniter. CodeIgniter підтримує популярну архітектуру MVC (Model-View-Controller), яка допомагає організувати код, розділяючи його на три основні компоненти: модель, представлення та контролер. Це робить код більш читабельним, спрощує його підтримку та модифікацію. CodeIgniter було обрано завдяки його зручності, швидкості та легкій архітектурі, що дозволяє уникнути розробки "з нуля". Bootstrap використовувався для спрощення фронтенд-розробки. У якості системи

управління базою даних було обрано MySQL. Доступ до системи здійснюється через стандартні веб-браузери.

3.6.2. Функціональний аналіз інтерфейсу менеджера проекту

Інтерфейс системи, доступний виключно для PM (рисунок 3.8), організовано за допомогою меню, що надає доступ до широкого спектра функцій.

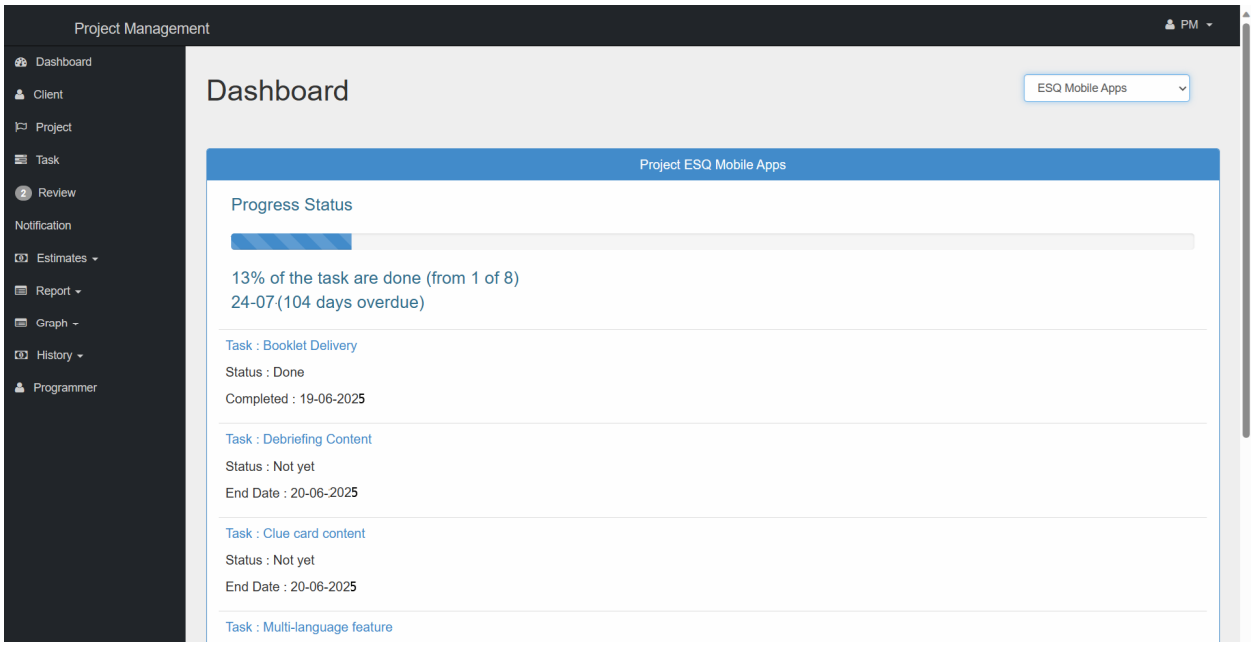

Рис. 3.8. Дашборд для менеджера проекту

За допомогою дашборда PM має можливість налаштовувати наступні елементи:

- Панель управління (Dashboard): Надає огляд статусу проекту.

- Управління клієнтами та проектами: Дозволяє керувати інформацією про клієнтів та проектами.

- Управління завданнями: Дозволяє PM переглядати та керувати завданнями та їх статусом.

- Звіти та аналіз: Забезпечує можливості для генерації звітів та візуалізації даних.

3.6.3. Деталізований розгляд ключових функцій

Управління проектами та завданнями

На сторінці "Проект" (рисунок 3.9) PM може переглядати, додавати, редагувати та видаляти проекти. Інтуїтивно зрозумілий інтерфейс дозволяє ефективно керувати списком проектів. Аналогічно, сторінка "Завдання" (рисунок 3.10) надає табличне представлення завдань, що полегшує моніторинг їх прогресу та відповідальних виконавців.

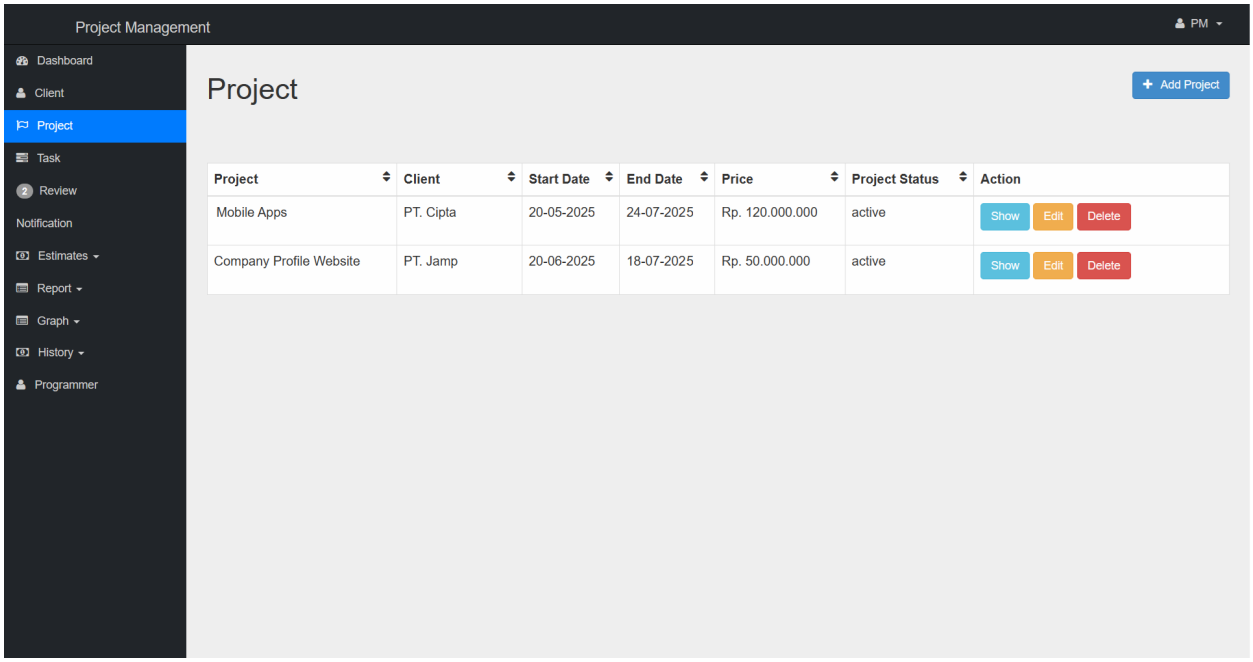

Рис. 3.9. Сторінка проектів

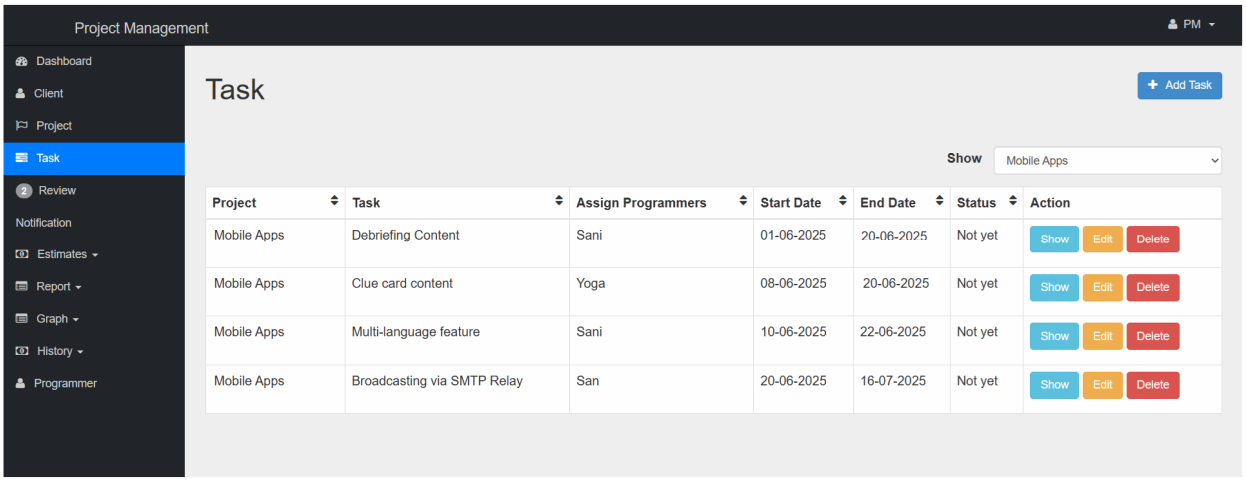

Рис. 3.10. Сторінка із задачами певного проекту

Оцінка та аналітика

Система надає інструменти для фінансового та часового контролю. На сторінці "Звіт про оцінку" (рисунок 3.11) PM може порівнювати планові та фактичні витрати. Функція генерації PDF-звітів (рисунок 3.12) забезпечує можливість документування та поширення цієї інформації.

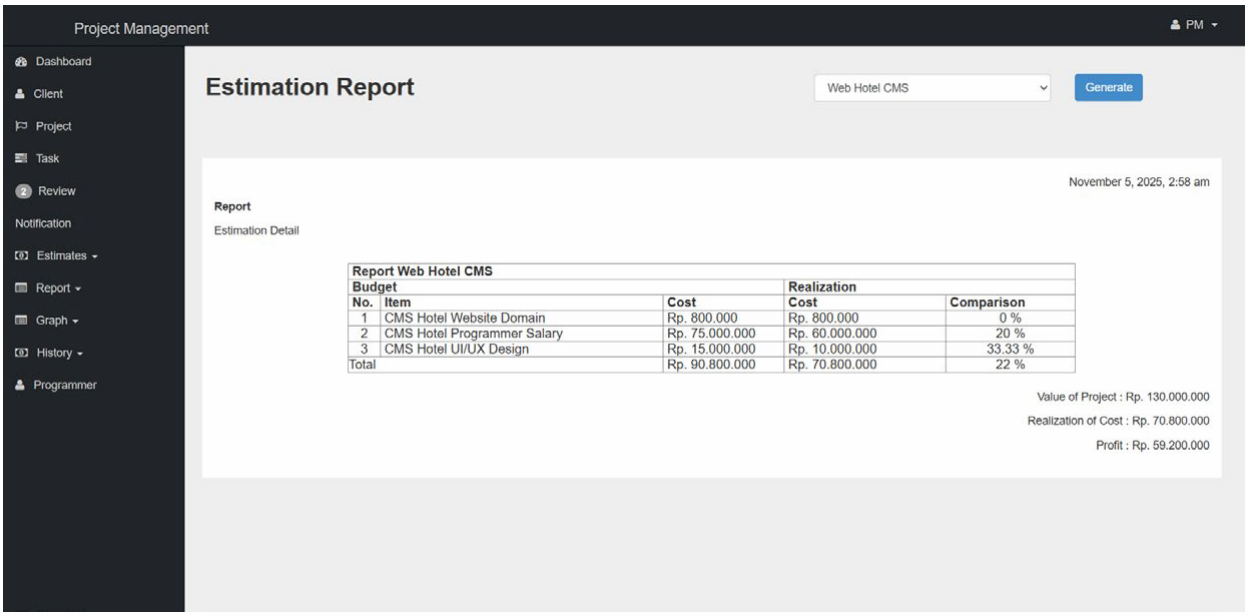

Рис. 3.11. Вигляд звіту про оцінку

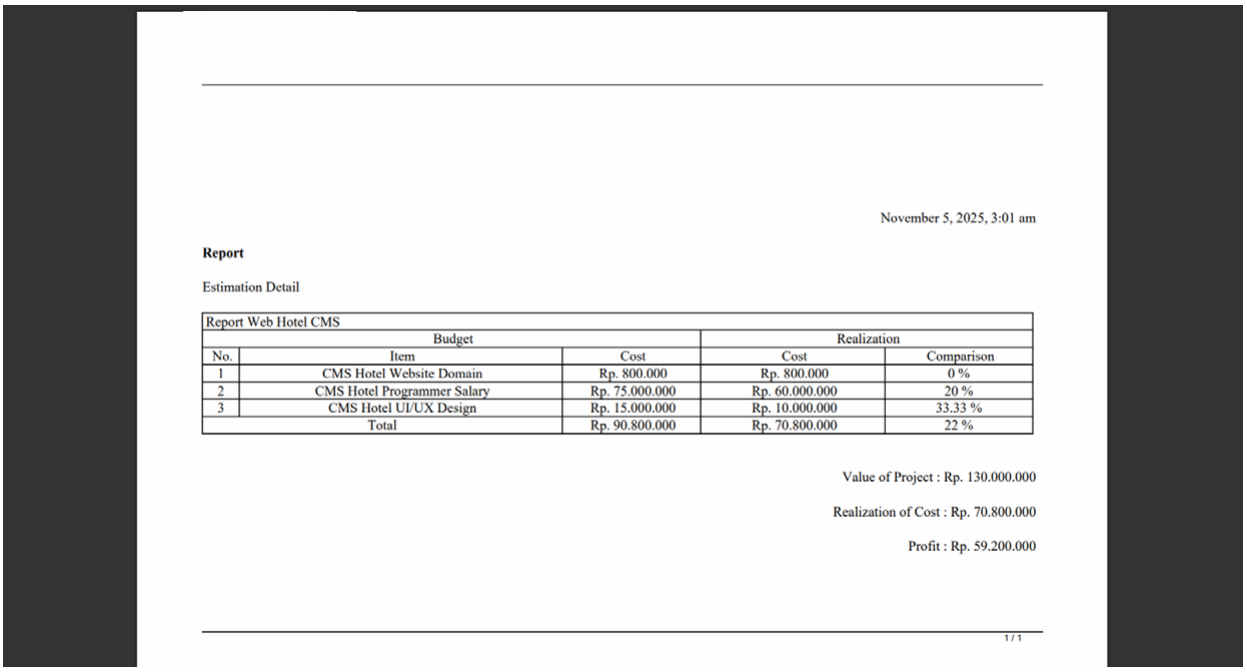

Рис. 3.12. Згенерований PDF звіту

Візуалізація даних

Функціонал візуалізації (рисунок 3.13) дозволяє PM швидко та ефективно аналізувати дані. Графічне порівняння планових та фактичних показників допомагає виявляти відхилення та вживати проактивних заходів. Графіки, такі як діаграма оцінки (рисунок 3.14) та діаграма реалізації часу (Рисунок 9), надають візуальне представлення прогресу проекту, що є критично важливим для прийняття обґрунтованих рішень.

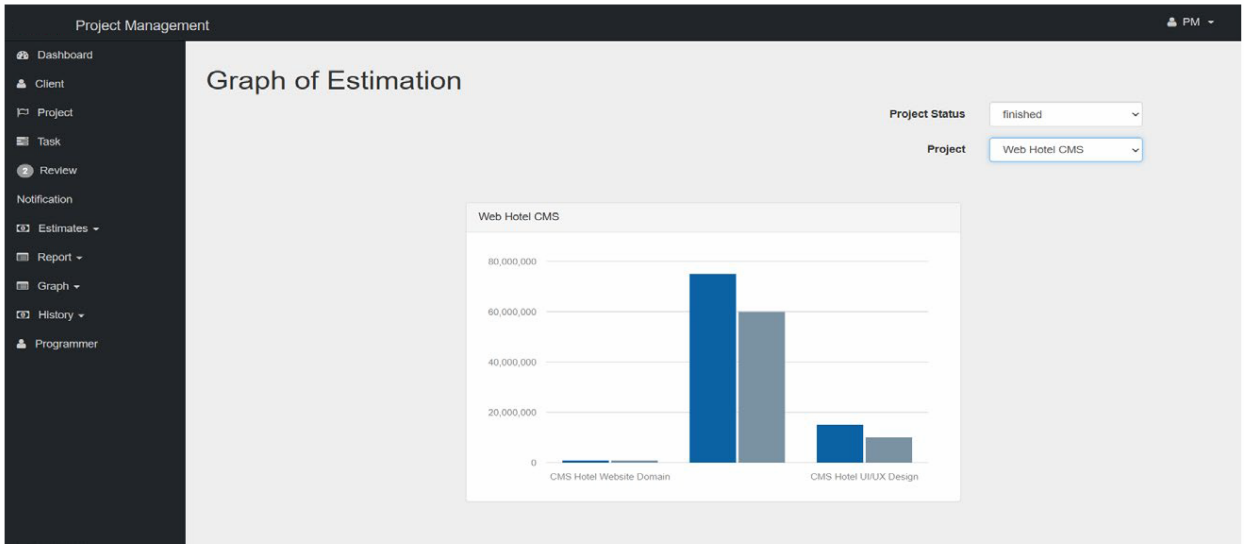

Рис. 3.13. Графічне представлення оцінки проекту

Рис. 3.14. Діаграма реалізації часу

На основі аналізу функціональності системи управління проектами в ІТ-компанії та результатів її тестування можна зробити висновок, що додаток ефективно виконує наступні ключові завдання:

1. Управління проектами.

Система забезпечує всебічний контроль над проектами, охоплюючи управління бюджетом, клієнтами, проектами та розподілом обсягу робіт на окремі завдання.

2. Планування.

Додаток надає інструменти для детального планування завдань та діяльності, що є необхідним для успішного завершення проектів.

3. Інформування та моніторинг.

Система виконує функцію інформування менеджерів проектів (PM) та програмістів про поточні завдання, а також відстежує їх прогрес. Це сприяє підвищенню прозорості та комунікації всередині команди.

4. Контроль витрат.

Завдяки можливості візуалізації даних, додаток дозволяє здійснювати моніторинг витрат, порівнюючи плановий бюджет з фактичними витратами, що допомагає вчасно виявляти відхилення та приймати обґрунтовані управлінські рішення.

Висновки до розділу

У третьому розділі було здійснено практичну реалізацію запропонованих моделей та методів управління програмними проєктами шляхом побудови системи планування та відстеження. Дослідження підтвердило доцільність інтеграції елементів Scrum та Extreme Programming у єдиний фреймворк, що забезпечує підтримку як стратегічного, так і операційного рівнів управління.

По-перше, проведено аналіз функціональних можливостей системи планування та розроблено архітектуру, що охоплює планування проєкту в

цілому, ведення беклогу, формування ітерацій, управління історіями користувачів та окремими завданнями. Такий підхід дозволив відобразити багаторівневу структуру робочих елементів, що відповідає сучасним вимогам до організації програмних проєктів.

По-друге, реалізовано інструменти відстеження виконання завдань, які охоплюють як загальний стан проєкту, так і прогрес окремих ітерацій, історій користувачів та технічних завдань. Це забезпечує прозорість процесу розробки та дозволяє оперативно виявляти відхилення від плану. Додаткові функції системи спрямовані на полегшення взаємодії між учасниками команди та надання менеджеру проєкту актуальної інформації для прийняття рішень.

По-третє, у ході дослідження створено системну модель фреймворку управління, яка поєднує методологічні засади гнучких підходів із практичною імплементацією у вигляді прототипу системи. Це дало можливість перевірити працездатність обраних рішень та оцінити їхню ефективність у контексті підтримки процесів управління програмними проєктами.

По-четверте, виконано проектування архітектури системи управління проєктами з детальним розглядом ключових функцій та аналізом інтерфейсу менеджера. Розроблений інтерфейс орієнтований на зручність використання та інтуїтивність взаємодії, що підвищує якість комунікації в команді та скорочує час на виконання управлінських операцій.

ВИСНОВКИ

У результаті виконання дослідження, присвяченого аналізу, моделюванню та імплементації методів і методологій систем планування та відстеження програмних проєктів, було досягнуто низку важливих наукових і практичних результатів.

По-перше, проведено детальний огляд предметної області застосування гнучких методологій управління проєктами, зокрема Scrum та Extreme Programming. У процесі дослідження було визначено їхні ключові характеристики, принципи функціонування, а також особливості інтеграції в гібридних підходах. На основі цього вдалося сформувати цілісне уявлення про еволюцію та сучасний стан гнучкої парадигми управління проєктами у сфері розробки програмного забезпечення.

По-друге, здійснено порівняльний аналіз основних методологій Agile, що дозволив виокремити сильні та слабкі сторони кожної з них. Особливу увагу приділено можливостям їх адаптації до специфіки програмних проєктів різного масштабу та складності. Важливим результатом стало формування теоретичної бази для розробки інтегрованих моделей планування й відстеження, які поєднують переваги Scrum та XP.

По-третє, було розглянуто методи оцінки та контролю процесів планування проєктів, у тому числі інструменти емпіричного підходу. Розроблено критерії оцінювання результативності систем управління, а також підготовлено набір тестових даних, що дозволив здійснити верифікацію запропонованих моделей. Це забезпечило можливість більш об’єктивної оцінки ефективності обраних методологій у реальних умовах.

По-четверте, створено системну модель та методологічні основи побудови фреймворку планування й відстеження програмних проєктів. Запропоновані підходи враховують потреби різних рівнів управління — від загального планування до контролю виконання окремих завдань та історій користувачів. Доведено, що інтеграція гнучких практик дозволяє підвищити

прозорість процесів, покращити комунікацію між учасниками команди та забезпечити більш високу адаптивність системи до змін вимог.

По-п’яте, виконано імплементацію функціональної системи управління програмними проєктами, яка включає модулі для планування, моніторингу та контролю виконання. Розроблена архітектура забезпечує підтримку ітеративних процесів, ефективне ведення беклогу, управління ітераціями, завданнями та історіями користувачів. Особливу увагу приділено інтерфейсу менеджера проєкту, що сприяє зручності роботи та підвищує якість управлінських рішень.

Таким чином, у роботі досягнуто комплексного результату, що полягає у поєднанні теоретичного обґрунтування та практичної реалізації моделей і методів управління програмними проєктами. Запропоновані підходи можуть бути використані як у наукових дослідженнях для подальшого розвитку гнучких методологій, так і в практичній діяльності ІТ-компаній для підвищення ефективності управління програмними продуктами.

Основними науковими результатами є:

- узагальнення та систематизація знань про гнучкі методології управління програмними проєктами;

- розробка інтегрованої моделі планування та відстеження програмних проєктів;

- формування критеріїв оцінки ефективності систем управління проєктами;

- створення концептуальної архітектури та прототипу системи управління проєктами.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Sikora, M. (2021). Factors influencing the improvement of procurement processes and cooperation with suppliers. European Research Studies Journal, 24(1), 1129-1138
2. Beck, K. (1999). Extreme Programming Explained: Embrace Change. Addison-Wesley.
3. Boehm, B. W., & Turner, R. (2004). Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley.
4. Cohn, M. (2009). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley.
5. Larman, C., & Basili, V. (2003). Iterative and Incremental Development: A Brief History. IEEE Computer.
6. Larman, C., & Vodde, B. (2010). Practices for Scaling Lean & Agile Development: Successful Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Addison-Wesley.
7. Highsmith, J. (2002). Agile Software Development Ecosystems. Addison-Wesley.
8. Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. IEEE Computer.
9. Pressman, R. S. (2010). Software Engineering: A Practitioner's Approach. McGraw-Hill.
10. Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley.
11. Rumbaugh, J., Jacobson, I., & Booch, G. (2005). The Unified Modeling Language Reference Manual. Addison-Wesley.
12. Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. Addison-Wesley.
13. Schwaber, K. (2004). Agile Project Management with Scrum. Microsoft Press.

14. Schwaber, K., & Beedle, M. (2002). Agile Software Development with Scrum. Prentice Hall.
15. Stellman, A., & Greene, J. (2014). Learning Agile: Understanding Scrum, XP, Kanban, and More. O'Reilly Media.
16. Cohn, M. (2010). The Agile Leader: How to Lead Your Team Through Change. Apress.
17. Lyytinen, K. (1987). Different perspectives on information systems: Problems and solutions. Association for Computing Machinery Computing Surveys (CSUR), 19(1), 5-46.
18. Leffingwell, D. (2007). Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley.
19. Leffingwell, D., & Widrig, D. (2003). Managing Software Requirements: A Unified Approach. Addison-Wesley.
20. Ghezzi, C., Mandrioli, D., & Morasca, S. (2003). A Formal Framework for Software Process Modeling and Analysis. IEEE Transactions on Software Engineering.
21. Vitharana, P., & Dissanayake, P. (2018). A Conceptual Framework for Project Management in Sri Lankan Construction Industry. International Journal of Scientific and Research Publications.
22. Pinto, J. K., & Slevin, D. P. (1988). Critical Success Factors in Effective Project Management. Project Management Journal.
23. Cleland, D. I. (2007). Project Management: Strategic Design and Implementation. McGraw-Hill.
24. Müller, R.; and Turner, R. (2007). The influence of project managers on project success criteria and project success by type of project. European management journal, 25(4), 298-309.
25. Project Management Institute (2017). A Guide to the Project Management Body of Knowledge (PMBOK® Guide). PMI.
26. Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Addison-Wesley.

27. Highsmith, J. (2009). Agile Project Management: Creating Innovative Products. Addison-Wesley.
28. Auramo, Jaana & Kauremaa, Jouni & Tanskanen, Kari. (2005). Benefits of IT in supply chain management–an explorative study of progressive Finnish Companies. Internaional Journal of Physical Distribution & Logistics Management, 35.
29. D'Hondt, M., & D'Hondt, T. (2007). The Agile Manifesto in Context: A Historical and Philosophical Perspective. In Proceedings of the 2nd International Conference on Agile Processes and Practices in Software Engineering.
30. Al-Hassan, M. A. (2012). A Comparative Study of Agile and Traditional Software Development Methodologies. In Proceedings of the 3rd International Conference on Computer Science and Information Technology.
31. Fowler, M. (2005). Agile Software Development. Communications of the ACM.
32. Tariq, S.; Ahmad, N.; Ashraf, M.U.; Alghamdi, A.M.; and Alfakeeh, A.S. (2020). Measuring the impact of scope changes on project plan using EVM. IEEE Access, 8, 154589-154613.
33. Cockburn, A. (2006). Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley.
34. De Luc, J., & Coad, P. (1997). Feature-Driven Development: A Framework for Agile Software Engineering. In Proceedings of the 1st International Conference on Agile Software Development.
35. Larman, C. (2003). Agile Software Development: Lessons Learned from a Large-Scale Project. IEEE Software.
36. Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. IEEE Computer.
37. Chardine-Baumann, E., & Botta-Genoulaz, V. (2014). A framework for sustainable performance assessment of supply chain management practices. Computers & Industrial Engineering, 76, 138–147.

38. Kaur, H., Singh, S. P., Garza-Reyes, J. A., & Mishra, N. (2018). Sustainable stochastic production and procurement problem for resilient supply chain. Computers & Industrial Engineering.
39. Royce, W. W. (1970). Managing the Development of Large Software Systems. In Proceedings of IEEE WESCON.

ДОДАТКИ

Додаток А

Фрагменти програмних кодів

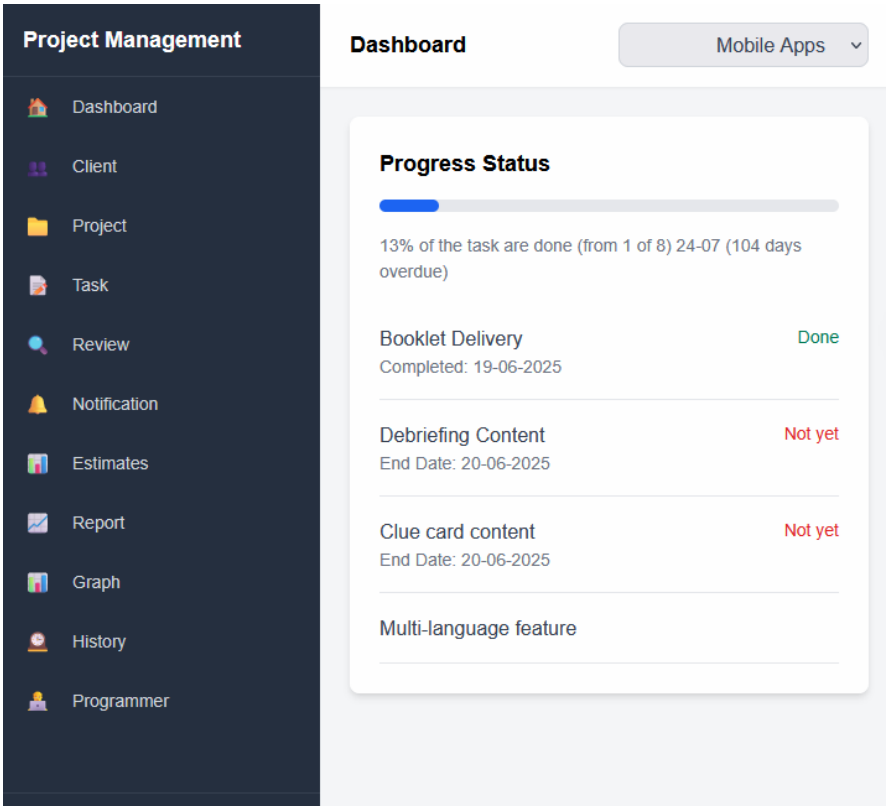

Рис. А.1. Мобільне представлення панелі менеджера проектів

```
import { useState };

const SidebarIcon = ({ children }: { children: React.ReactNode }) => (
  <div className="w-6 h-6 flex items-center justify-center text-gray-
400">{children}</div>
);

const SidebarItem = ({ icon, text }: { icon: React.ReactNode; text:
string }) => (
  <div  className="flex  items-center  py-3  px-4  hover:bg-gray-700
cursor-pointer">
    {icon}
    <span className="ml-4 text-gray-300 text-sm">{text}</span>
  </div>
);
```

```
const TaskItem = ({ task, status, completedDate, endDate }: { task: string; status: string; completedDate?: string; endDate?: string }) => (
  <div className="py-4 border-b border-gray-200">
    <div className="flex justify-between">
      <span className="text-gray-700">{task}</span>
      <span className={`text-sm ${status === 'Done' ? 'text-green-600' : 'text-red-600'}`}>{status}</span>
    </div>
    {completedDate && (
      <div         className="text-sm         text-gray-500">Completed: {completedDate}</div>
    )}
    {endDate && (
      <div className="text-sm text-gray-500">End Date: {endDate}</div>
    )}
  </div>
);

export default function Dashboard() {
  const [project, setProject] = useState('ESG Mobile Apps');

  return (
    <div className="flex h-screen bg-gray-100">
      <div className="w-64 bg-gray-800 text-white flex flex-col">
        <div className="py-4 px-4 border-b border-gray-700">
          <span          className="text-lg          font-bold">Project Management</span>
        </div>
        <div className="flex-1 overflow-y-auto">
          <SidebarItem             icon={<SidebarIcon></SidebarIcon>} text="Dashboard" />
          <SidebarItem             icon={<SidebarIcon></SidebarIcon>} text="Client" />
          <SidebarItem             icon={<SidebarIcon></SidebarIcon>} text="Project" />
          <SidebarItem icon={<SidebarIcon></SidebarIcon>} text="Task" />
```

```
          <SidebarItem              icon={<SidebarIcon></SidebarIcon>}
text="Review" />
          <SidebarItem              icon={<SidebarIcon></SidebarIcon>}
text="Notification" />
          <SidebarItem              icon={<SidebarIcon></SidebarIcon>}
text="Estimates" />
          <SidebarItem              icon={<SidebarIcon></SidebarIcon>}
text="Report" />
          <SidebarItem              icon={<SidebarIcon></SidebarIcon>}
text="Graph" />
          <SidebarItem             icon={<SidebarIcon></SidebarIcon>}
text="History" />
          <SidebarItem             icon={<SidebarIcon></SidebarIcon>}
text="Programmer" />
        </div>
        <div className="py-4 px-4 border-t border-gray-700">
          <span className="text-sm text-gray-400">PM</span>
        </div>
      </div>
      <div className="flex-1 flex flex-col overflow-y-auto">
        <div  className="bg-white  shadow-sm  py-4  px-6  flex  justify-
between items-center">
          <span className="text-lg font-bold">Dashboard</span>
          <div className="flex items-center">
            <select
              value={project}
              onChange={(e) => setProject(e.target.value)}
              className="py-2 pl-10 pr-4 text-gray-700 border border-
gray-300 rounded-lg"
            >
              <option value="ESG Mobile Apps">ESG Mobile Apps</option>
            </select>
          </div>
        </div>
        <div className="p-6 flex-1 overflow-y-auto">
          <div className="bg-white rounded-lg shadow-md p-6">
            <div    className="text-lg    font-bold    mb-4">Progress
Status</div>
```

```
            <div className="w-full bg-gray-200 rounded-full h-2.5 mb-
4">
              <div className="bg-blue-600 h-2.5 rounded-full" style={{
width: '13%' }}></div>
            </div>
            <div  className="text-sm  text-gray-500  mb-4">13%  of  the
task are done (from 1 of 8) 24-07 (104 days overdue)</div>
            <TaskItem     task="Booklet     Delivery"     status="Done"
completedDate="19-06-2025" />
            <TaskItem   task="Debriefing   Content"   status="Not   yet"
endDate="20-06-2025" />
            <TaskItem   task="Clue   card   content"   status="Not   yet"
endDate="20-06-2025" />
            <TaskItem task="Multi-language feature" status="" />
          </div>
        </div>
      </div>
    </div>
  );
}
```