НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий Інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:

Завідувач кафедри
_____ В.Є. Мухін
«___»_____2022 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»
на тему: «Аналіз і створення засобів управління потоками робіт як інструменту менеджера проектів»

Виконав:
студент 4 курсу, групи ДА-82
Ліщенко В’ячеслав Владиславович _____

Керівник:
доцент, к.т.н., с.н.с.
Кисельов Г.Д. _____

Консультант з нормоконтролю:
доцент, к.т.н. Кирюша Б.А. _____

Рецензент:
доцент кафедри АПЕПС ТЕФ,
доцент, к.т.н. Шаповалова С.І. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий Інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.Є. Мухін
«___»_____2022 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ліщенку В'ячеславу Владиславовичу

1. Тема роботи «Аналіз і створення засобів управління потоками робіт як інструменту менеджера проектів», керівник роботи Кисельов Генадій Дмитрович, к.т.н., затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи

3. Вихідні дані до роботи: Trello, Jira, Asana, Каскадна модель, Ітераційна модель, V-модель, Спіральна модель, kanban, agile, scrum, extreme programming, аналіз та оцінка вартості сучасних рішень управління проектами.

4. Зміст роботи

- Методи контролю і планування операцій, які здійснюються в системах
- Управління їх вартістю і користю в умовах безлічі користувачів
- Огляд сучасних концепцій управління IT проектами.
- Рекомендації що до створення власного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1) Презентація до захисту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., к.е.н., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Збір інформації	02.05-08.05 2022	
2	Аналіз методів контролю і планування операцій	08.05.2022- 12.05	
3	Аналіз управлінням вартістю і користю в умовах безлічі користувачів	12.05-17.05.2022	

4	Огляд сучасних концепцій управління IT проектами.	17.05-23.05.2022	
5	Порівняння продуктивності проаналізованих компонентів	26.05-28.05.2022	
6	Розробка рекомендацій, що до створення власного продукту	28.05-06.06.2022	
7	Оформлення дипломної роботи	16.06.2022	
8	Отримання допуску до захисту та подача роботи в ДЕК	20.06.2022	

Студент Ліщенко В.В.

Керівник Кисельов Г.Д.

АНОТАЦІЯ

Метою моєї дипломної роботи було проведення аналізу засобів управління потоками робіт як інструменту менеджера проектів та надання рекоммендацій що до створення власного проекту. Зокрема веб-орієнтованого середовища «Управління ІТ проектами».

У просторі сучасного ринку існує занадто велика кільскість подібних рішень, які пропонують середовища управління ІТ проектами в залежності від потреб конкретного користувача або проекту. В межах цієї дипломної роботи було аналізовано та порівняно, які методи контролю і планування операцій здійснюються в системах. Було порівняно обидва традиційні та сучасні методи та розглянуто переваги та недоліки їх. Проведений аналіз систем управління вартістю і користю функціоналу в умовах безлічі користувачів.

В роботі зроблені рекомендації що до створення інструменту веб-орієнтованого середовища управління ІТ проектами а також самого середовища. Ціль цього веб-сервіса – покриття як умога більше функціоналу для забезпечення роботи компаній-гігантів ІТ простору, в умовах постійного розширення кількості користувачів. Головною задачою цієї роботи був аналіз сучасних концепцій управління ІТ проектами, дослідження їх актуальності, доцільнотсті та цінності на сучасному ринку.

В результаті проведенної роботи було узагальнено критерії, а також винесено вирок доцільності створення власного продукту, та рекомендовані алгоритми створення власного продукту.

Ключові слова: ІТ проекти, управління ІТ проектами, менеджмент Trello, Jira, Asana, Каскадна модель, Ітераційна модель, V-модель, Спіральна модель, kanban, agile, scrum, extreme programming, аналіз та оцінка вартості сучасних рішень управління проектами.

ANNOTATION

The purpose of my diploma work was carrying out and found in funds workflow management as a project manager tool and providing recommendations for creating your own project . In particular web -oriented IT Project Management environment .

In space There is a modern market too many _ similar solutions that offer environment in the management of IT projects depending on the need specific user or project . Within this diploma work was analyzed and compared , as well as control and planning operations carried out in systems. It was comparatively both traditional and modern methods and considered advantages and disadvantages them . The analysis of systems in management is carried out cost and benefits functional in the conditions many users .

At work made and recommendations for creating a web -based environment in IT project management and . Target of this web service - coverage as a condition of more functionality to ensure the work of IT giants, in a constantly expanding number of users . The main task this one work was anal of modern concepts of IT project management, research of their relevance, feasibility and value in today's market .

As a result conducted work the criteria were summarized, as well as the verdict of expediency of creating your own product, and recommended algorithms for creating your own product .

Key words: IT projects , IT project management , management Trello , Jira , Asana , Cascade model, Iterative model, V-model, Spiral model, kanban , agile , scrum , extreme programming , analysis and evaluation value modern decisions project management .

ЗМІСТ

ВСТУП

Інструменти управління проектами міцно увійшли в сучасне життя і стали невід'ємною частиною будь-яких продуктових команд: починаючи від IT-відділів великих корпорацій і закінчуючи численними початківцями.

Вже протягом кількох десятиліть задача визначення підходів та способів підвищення продуктивності, якості та надійності розробки ПЗ є актуальною. При цьому, кожен фахівець має свої власні погляди щодо покращення процесу, який є важко передбачуваним. В той час, як одні намагаються вирішити питання шляхом систематизації та формалізації, другі – застосувати методи управління проектами та методи програмної інженерії, а треті – дотримуються думки, що розробка продукту повинна підлягати постійному контролю замовника . До цього ж, методологія розробки ПЗ повинна бути порівняна із складністю структури самого ПЗ. Інакше, якщо обрана методологія є невиправдано складною для даного продукту, то це лише збільшить вартість розробки.

Життєвий цикл ПЗ представляє собою стадії, через які проходить ПЗ від появи ідеї до його подальшої підтримки та «смерті». Тобто, у ПЗ, подібно до живої істоти, є свій життєвий цикл, який багато в чому і зумовлює методології розробки ПЗ .

Принципи, поняття, ідеї, методи, способи та засоби – вся ця сукупність визначає методологію розробки, а своєчасна оцінка вартості і користі не дає проекту “померти”.

У своїй дипломній робота я розглядаю існуючі комерційні та безкоштовні рішення крнтролю розробки ПЗ і провести іх аналіз; методи контролю і планування операцій які здійснюються в системах; методи встановлення цілей проектів, управління їхньою вартістю і користю в умовах безлічі користувачів.

1 КОНТРОЛЬ І ПЛАНУВАННЯ ПРОЦЕСІВ ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

1.1 Огляд Agile технологій

В Сьогоденні інформаційні технології проникли і поринули в усі сфери нашого життя, оскільки в світі спостерігається зосередження на цифровізації, а з нею приходить й усвідомлення важливості ПЗ. Необхідність підвищення швидкості реагування і покращення можливостей обслуговування все більшої кількості клієнтських потреб. Тиск, який дедалі більше відчувають на собі компанії, що займаються розробкою ПЗ призводить не лише до необхідності привнесення деякої гнучкості бізнесу з метою впровадження нової продукції на біржу, але й до необхідності модифікувати вже існуючі рішення та послуги .

Нові технології, пропозиції та можливості конкурентних фірм, прийняті нові закони, загальні коливання ринку, роблять актуальним існування арен для конкурентної боротьби пов’язаної із швидким постачанням ПЗ підприємством. Переможцем із цього поєдинку може вийти лише той, хто не боїться змін, готовий оновлювати свою діяльності, спрямований на керування розробкою, зокрема, на безліч формальних і неформальних процедур, практик, процесів і правил . Ці механізми керування - важлива функція в управлінні та контролю над тим, як ПЗ постачається у виробництво.

Продуктивність індивідуумів та команд, час виходу на біржу проектів, зрілість процесу в послідовності, уніфікації та стандартизації практик та якість коду, оброблення помилок та обслуговування запитів – чотири ключові виміри, отримання балансу у яких є традиційним завдання для організацій з постачання ПЗ.

Віднайти “золоту” середину все важче із плином часу. Отже, відбувається нагромадження тиску з отримання все більшої швидкості реагування. Створені виклики спрямовують організації до неминучого впровадження гнучких методів у політику організації. Дослідження свідчать про те, що гнучкі практики - домінуючі підходи у модернових організаціях розробки ПЗ.

Метод гнучкості відкриває нові можливості в галузі забезпечення якості ПЗ, оскільки це не лише задоволення потреб клієнтів, але й можливість змінювати вимоги аж до рівня розгортання продукту .

Гнучкі технології розробки ПЗ дозволяють підтримувати постійний зворотній зв'язок і враховувати зміни у вимогах протягом всього життєвого циклу розробки ПЗ, підтримувати тісну співпрацю між клієнтами і розробниками, а також забезпечувати ранні і часті випуски програмних функцій, необхідних для системи(рис. 1.1).

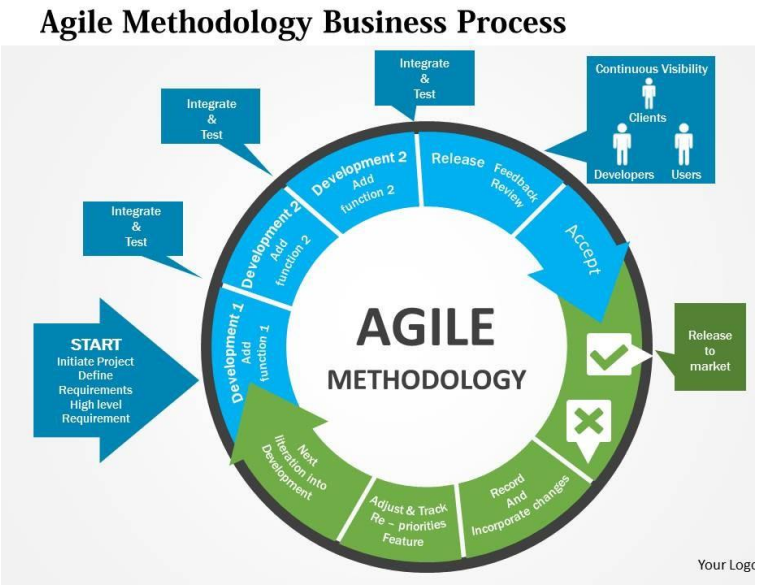


Рисунок 1.1 - Agile методологія

Маніфест Agile – основа, на якій побудовані методи гнучких технологій розробки ПЗ. Він був опублікований групою розробників ПЗ та консультантів у 2001 році.

Відповідно до маніфесту Agile :

Ми постійно відкриваємо для себе більш досконалі методи розробки ПЗ, займаючись розробкою безпосередньо і допомагаючи в цьому іншим. Завдяки виконаній роботі ми змогли усвідомити, що :

Люди і взаємодія *важливіше процесів та інструментів*

Працюючий продукт *важливіше вичерпної документації*

Співпраця з замовником *важливіше узгодження умов контракту*

Готовність до змін *важливіше проходження попереднім планом*

Тобто, не заперечуючи важливості того, що справа, ми все-таки більше цінуємо те, що зліва .

Гнучкі методології можна визначити як групу процесів розробки ПЗ(рис. 1.2)

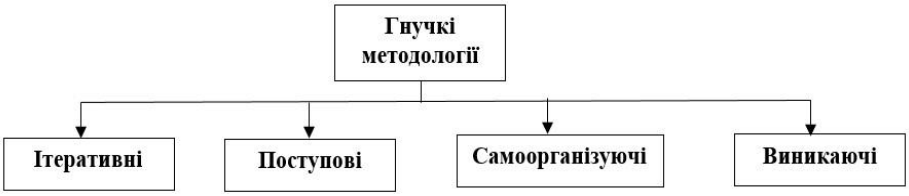


Рисунок 1.2 - Визначення гнучких методологій

- ***ітераційними*** як спроба вирішення проблем шляхом пошуку послідовних наближень до рішення.
- ***поступовими*** як підхід, який передбачає розподіл системи на функціональні підсистеми та розробку оновленої функціональності до загальної системи з кожним випуском;
- ***самоорганізуючими*** як концепція, що дає команді можливість організовувати самостійно процес для найкращого результату завдання

- ***виникаючими*** як досвід навчання з кожного проекту, внаслідок того, що кожен проект організовується по-різному, застосовуючи ітеративні, поступові, самоорганізуючі та новітні методи.

Деякі з відомих методів гнучких технологій розробки ПЗ - екстремальне програмування (XP), Scrum, кристальні методології Crystal Clear, Kanban, розробка керована функціями, динамічний метод розробки .

1.1.1 Scrum

Scrum – один із головних сучасних методів гнучких технологій з розробки ПЗ, який є ітераційним та інкрементним підходом до керування проектами. В проектах розробки ПЗ із Scrum є основними такі ролі:

Перша роль - це *Scrum-майстер*. Це людина яка допомагає команді правильно використовувати ітерації Scrum для виконання процесів на найвищому рівні. Від звичайного менеджера проекту його відрізняють ролі, які не забезпечує щоденне керівництво командою і не призначають завдання окремим особам.

Власник продукту представляє бізнес, клієнтів, користувачів та скеровує команду до правильної мети, аби створити якісний продукт. Виконується ця частина за допомогою створення бачення кінцевого продукту із подальшою передачею цього образу команді.

Третя і остання роль – команда. Оскільки людина, яка вирішує хто що буде робити відсутня, то така група повинна бути саморганізовувною. Питання по розподілу задач і зайнятості вирішуються командою в цілому на зустрічах. Scrum-команда є функціональною, тобто кожен повинен взяти участь в розробці, від ідеї до реалізації .

Для того, аби уявити процес зв’язку зазначених ролей, можна собі уявити спортивний автомобіль. Нехай команда буде - сам автомобіль, який готовий до

руху у будь-якому із напрямків. Водієм - власник продукту, в обов'язки якого входить керування в правильному напрямку. Головним механіком в даному процесі є Scrum-майстер, який підтримує автомобіль у гарному стані.

Якщо говорити про артефакти, то кожному проекту Scrum властиві такі п'ять

.

Продукт - первинний артефакт, який приводиться командою до потенційно доступного стану в кінці кожного спринту(Sprint).

Відставання продукту - це повний перелік функціональних можливостей, які треба додати до продукту. Команда завжди працює над найціннішими з них згідно з пріоритезацією наданою власником.

Історія користувача або Шлях користувача – короткий опис функціональності з точки зору та поведінки користувача або клієнта під час використання продукту.

Спринтове відставання – артефакт, який являє собою певні завдання для виконання командою, щоб забезпечити функціональність, призначену на певинний спринт.

Як додатковий артефакт виступають *діаграми спринтового розгортання* і *діаграми випуску релізів*. Завдяки них можна відобразити обсяг виконаної та залишившоїся роботи.

Отже "визначені та повторювані процеси працюють лише для вирішення визначених та повторюваних проблем із визначеними та повторюваними людьми у визначених та повторюваних середовищах", а це, як очевидно, неможливо. Тому, для вирішення цієї проблеми процесів, проект розбивається на ітерації (спринти, Sprint).

Тобто, за даної моделі, проекти розробляються «покроково» через визначену серію спринтів (тривалістю 3-4 тижні) . Перед початком кожного нового спринту збирається мітинг або зустріч за для пріоритезації шляхів користувачів.

Протягом спринту команда із відібраної сукупності бере невелику кількість функцій і реалізує їх від стадії ідеї, до кодованої та перевіреної функціональності.

Кожного дня Scrum-команда проводить коротку зустріч (близько 15 хвилин), щоб кожен член команди міг відповісти на три головних запитання: «Що я зробив вчора, що я буду робити сьогодні, і які перешкоди стали на шляху вирішення завдання?» .

Наприкінці кожної ітерації здійснюється огляд спринту, метою якого є демонстрація нової функціональності. Зрештою, всі розроблені та прийняті функції інтегруються в ПЗ, що розробляється.

Зустріч пріоритезації завдан, щоденна зустріч команди, огляд спринту – контур зворотного зв'язку, який може призвести до змін у функціональності продукту, або до перегляду та додавання нових елементів до відставання продукту.

В кінці кожного спринту вся команда приймає участь у ретроспективі, щоб поміркувати над завершеним спринтом і визначити аспекти для покращення.

Scrum-процес ділиться декілька частин(рис. 1.3)

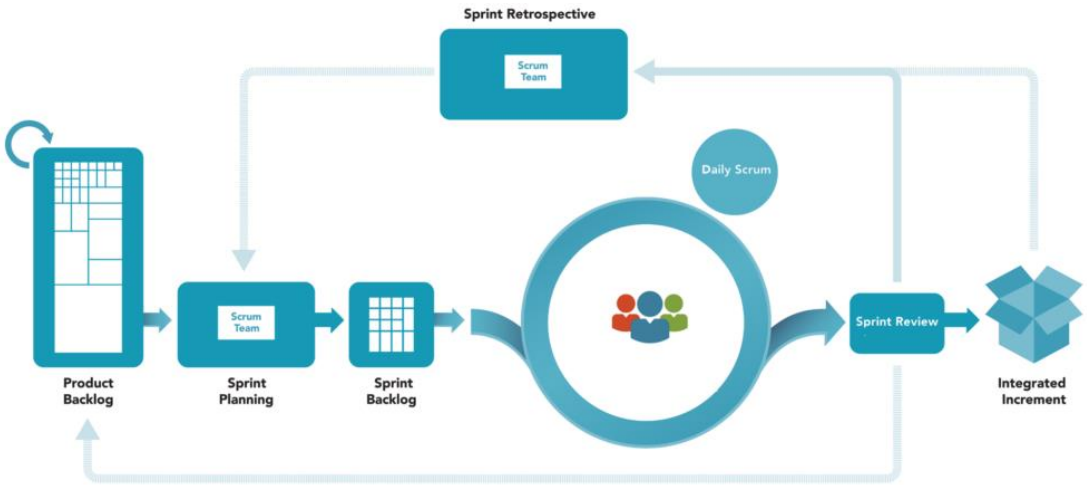


Рисунок 1.3 - Структура процесу Scrum

Згідно структурі, процес починається з етапу вимог. Вони збираються та аналізуються, що дає змогу сформувати задачі до відставання продукту. Відставання продукту пріоритезує найцінніші елементи зверху списку при обліку залежностей, елементи з більшим пріоритетом будуть реалізовані в найближчому спринті.

Реалізація системи слідує етапу аналізу і займає час в середньому 2-4 тижні, тобто ітерацію. Спринт починається з розподілу розробниками задач з верхньої частини відставання продукту і подрібнення їх на завдання, які вони завершують протягом спринту.

Завершуються спринти оглядом результатів ітерації власником ПЗ. Дані результати оцінюються і визначається чи були виконані цілі, встановлені на початку спринту. Цей етап також супроводжується навчальною сесією, що підкріплюється досвідом, з метою поліпшення практики роботи. На цей час ПЗ має бути протестоване і готове.

1.1.2 Extreme programming

Іншим відомим методом є *екстремальне програмування (XP)*. Як і інші методи гнучких технологій, він працює завдяки об’єднанню всієї команди методом простих практик із включенням достатнього зворотного зв’язку, виступає за коротку ітерацію і часті випуски робочого коду з метою підвищення продуктивності. XP зроблений для команд, що включають від 8 до10 учасників працюючих з об'єктноорієнтованою мовою програмування.

В XP процес починається зі створення історій користувача замовником, що описує функціональності і які можна реалізувати в середньому за 1-2 тижні роботи. Замовник, виходячи із кошторису, який йому надають, пріоритезує завдання. Реалізація функцій проходить ітераційно, що дозволяє кожні два тижні представляти результат роботи замовнику. Поступово система зростає у функціональності керуючись замовником на оглядах .

Нижче наведено основні практики, що використовуються в XP(рис. 1.4).

Вся команда – група куди входять замовник, програмісти та тестувальники , аналітики. Зазвичай, в команду входить ще і тренер та менеджер. Треба зазначити , що жодна з наведених ролей та представників не обов'язково виключно виконує свої прямі обов'язки та властивості, кожен у команді XP сприяє будь-яким способом створенню ПЗ.

Планування гри –тісна взаємодія клієнтської та виконавчої групи, яка дуже рекомендується з метою визначення пріоритетів й вимог для наступного випуску. Команда постачає клієнтові виключно узгоджені з замовником історії користувачів.

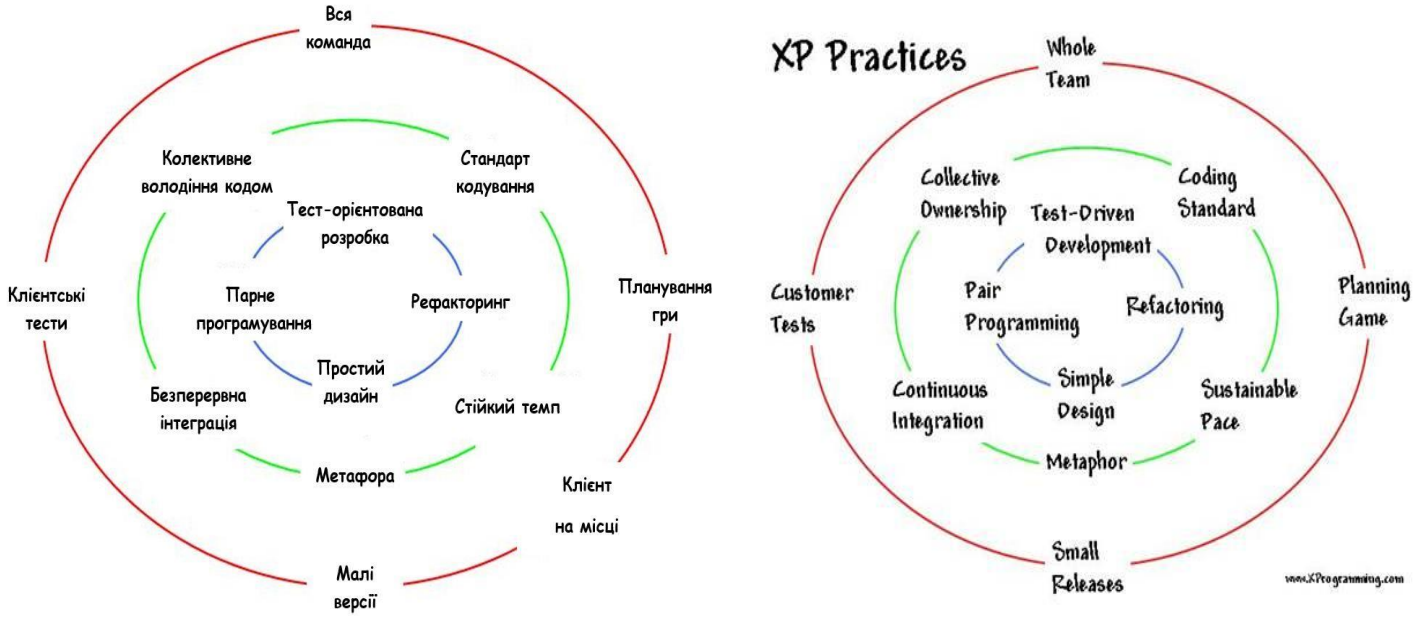


Рисунок 1.4 - Методологія екстремального програмування

Малі версії – демо-версії системи після декількох ітерацій.

Клієнтські тести- визначення клієнтом кількох автоматизованих приймальних тестів для кожної функції для демонстрації її працездатності . Наведені тести будуються і використовуються командою з метою приведення та перевірки коректності роботи ПЗ.

Простий дизайн – методологія заохочення розробників до максимально спрощеного адекватного дизайну системи, який підходить для функціональності системи .

Парне програмування – методика написання алгоритму двома програмістами, які сидять поряд. Це дає змогу та впевненість, що коди переглядаються не однією людиною, що призводить до ліпшого дизайну, процесу тестування та поліпшення алгоритму .

Тест-орієнтована розробка – методика, що передбачає критерій test-first і означає, що розробники пишуть приймальні тести, перш ніж писати сам код, а клієнти, пишуть функціональні тести для регресії кожної ітерації.

Рефакторинг – методика, що пропонує на початку проекту розробку банального дизайну із поступовим покращенням. Подібна методика розглядає розвиток системи через перетворення існуючої, так, щоб всі тести успішно виконалися.

Безперервна інтеграція – методика, за якої система зберігається повністю інтегрованою і весь новий алгоритм мерджиться в основу якомога частіше. Після кожного мерджу проводиться регресія по функціональним тестам.

Колективне володіння кодом – методика, коли всі програмісти володіють спільним алгоритмом і кожен може змінити/покращити якусь його частину одночасно з усіма.

Стандарт кодування – методика, за якої весь алгоритм дотриманий згідно загального стандарту кодування і увесь написаний код дотриманий одного стилю написання.

Метафора– методика, що передбачає розробку комплекту метафор (емоційних характеристик роботи ПЗ) группой для моделювання розроблюваної системи. Використовується загальний простір імен, за допомогою якої всі спілкуються і розуміють одне одного.

Стійкий ритм– методика, завдяки якій группа сама визначає ритм роботи і може працювати понаднормово, коли вважає це ефективним .

Клієнт на місці – методика, коли клієнт протягом всього часу розробки перебуває з командою і відповідає на виникаючі питання, забезпечуючи прогрес розробки.

1.1.3 Kanban

Kanban - це спосіб гнучких технологій, який допомагає здійснювати процеси проектування, управління та вдосконалення потокових систем. Його назва походить від того, що в основі нього лежить використання канбан-механізмів візуальної сигналізації, які дають змогу контролювати готовність для ПЗ. Він дотримується принципу, що робота безперервно протікає через систему, а не організовується у різні часові границі .

Даний алгоритм використовується у ситуаціях, де необхідна організація знань, коли робота є непередбачуваною та коли механізм роботи передбачає інкрементне представлення продукту замовнику.

Kanban команди дотримуються таких цінностей:

• прозорість - обмін інформацією покращує потік ділової цінності;

• баланс – збалансування різних аспектів, точок зору для досягнення єфективності;

• співпраця - покращення способу спільної праці;

• орієнтація на клієнта – напрямок оптимізації потоку вартості для споживачів, які є зовнішніми від системи;

• потік - робота яка є безперервною у одній часовій рамці;

• лідерство - здатність надихати інших діяти на всіх рівнях розробки;

• розуміння – пізнання (як індивідуальне, так і організаційне) вихідної позиції необхідно для просування вперед та вдосконалення;

• угода – кожен з команди спільно рухається до цілей, поважаючи та враховуючи розбіжності в думках та підходах;

• повага – цінування та розуміння у команді.

Такі методики є діяльністю, необхідною для управління системою Канбан (табл.1).

Таблиця 1 Практики Kanban

Практика	**Опис**
Візуалізація	Дошка Канбан як візуалізатор виконаної роботи та робочого процесу.
Обмеження	Злагодження потоку робіт, скорочення термінів виконання, поліпшення якості через обмеження обсягу виконуваної роботи в системі.
Управління потоком	Мінімізація часу виконання і максимальне передбачення – завдання потоку роботи. Ключовий аспект – виявлення та вирішення вузьких місць та «стін».
Чіткі правила	Обмеження WIP, розподіл потужності, визначення зробленого та інші правила для робочих завдань на різних етапах процесу.
Цикли зворотного зв'язку	Огляд стратегії, огляд ризиків, огляд операцій, огляд надання послуг, зустріч у Канбані, зустріч з планування доставки.
Спільне поліпшення	Застосування постійного та поступового вдосконалення початкового процесу розробки ПЗ.

Дуже відомою методикою Kanban є дошка, яка є обов’язковим елементом. Вона завжди у вільному доступі,щоб кожен член команди у будь-який час міг бачити на якому етапі перебуває його завдання або загальне.

Kanban дошка підійде будь-яка: можна використовувати просту коркову або програми-трекери на Trello, Jira, оскільки вона підлаштовується під будь-який процес в будь-якій області з метою складяння списоку завдань.

Кожен проект має власний план робіт, який спочатку проходить фазу аналізу і декомпозується на відповідному етапі. Наприклад, для процесу створення IT-проекту етапи можуть бути такими(рис 1.5): *Необхідно зробити, В процесі, Тестується, Зроблено*. При цьому, імена стовпців можуть змінюватися, не порушуючи методології, важливо лише зберігати їхню послідовність, оскільки поток - це ключова цінність Kanban .

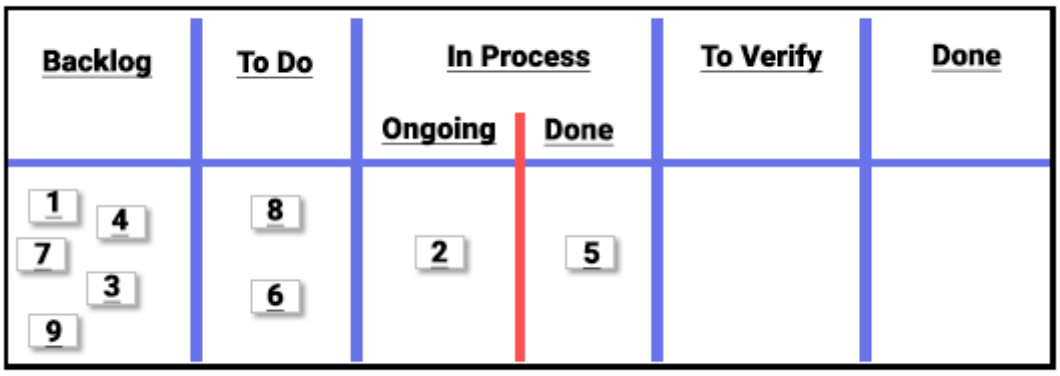


Рисунок 1.5 - Kanban дошка

Вся робота у Kanban - єдиний потік, де кожна система у процесі роботи відрізняється етапами. Отже найкращий спосіб описати життєвий цикл методу через задіяні петлі зворотного зв'язку(табл. 2).

Таблиця 2 Фази зворотного зв’язку

Фази зворотного зв’язку	**Частота проведення**	**Мета**
Огляд стратегії	поквартально	Визначення послуг, що надаються, та контекст доречності послуги.
Огляд надання послуг	1р на 2 тиж	Ретроспектива ефективності послуг, підвищення. якості
Огляд ризиків	щомісячно	Визначення ризиків доставки послуг.

Огляд операцій	щомісячно	Визначення балансу між послугами з метою максимізації доставки вартості.
Зустріч поповнення	щотижнево	Визначення предметів роботи команди.
Зустріч з планування доставки	згідно фази доставки	Відстеження та планування доставки споживачам.
Зустрч у Канбані	щоденно	Координація діяльності команди протягом дня.

Деякі команди ідейно об'єднують Scrum і Kanban у зручий Scrumban. Зі Scrum беруться спринти з фіксованою тривалістю і ролі, а з Kanban - концентрацію на тривалості циклу і обмеження кількості одночасно виконуваних завдань. Проте, на етапах адаптації та гнучкої трансформації рекомендується вибрати конкретну методику і деякий час слідувати виключно їй, адже, час для експериментів знайдеться завжди.

1.2 Огляд традиційних підходів

Коли програмування тільки зароджувалося, єдиними учасниками процесу розробки були найчастіше лише програмісти, які писали програмні функції для полегшення математичних розрахунків та автоматизації інших рутинних дій .

Сьогодні все інакше, сучасні системи досягливеличезних масштабів і складності, їхнє створення вимагає долучання до роботи цілих команд фахівців різного профілю: програмістів, аналітиків, системних адміністраторів, тестувальників і кінцевих користувачів. Через таку велику кількість осіб

виникає питання ускладненої координації. За для полегшення процесу координації, компанії дотримуються певних моделей життєвого циклу ПЗ.

Коли говорять про традиційні моделі життєвого циклу ПЗ, то найчастіше це (рис. 1.6)

Рисунок 1.6 - Класифікація моделей життєвого циклу ПЗ

Може здатися, що індустрія програмної інженерії прийшла, нарешті, до загальної , єдиної моделі. Але, каскадна модель, багаторазово розкритикована теорією і практикою. Ітераційна модель є яскравим представником еволюційного погляду, але є й єдиною моделю, яка приділяє явну увагу аналізу та попередження ризиків . На сьогоднішній час в різних джерелах наводиться списоки різних моделей та їхня інтерпретація. Нижче я представив виділені вище чотири моделі - ітераційну , спіральну, каскадну, V-модель, та.

1.2.1 Ітеративна розробка

Ітеративна модель розробки ПЗ – модель, яка передбачає поетапний процес, в якому кожна фаза додає функціональність до розроблюваного ПЗ та включає свій незалежний набір заходів з розробки та тестування . Дана модель, по суті, є перехідною (*від каскадної до Agile*) моделлю розробки ПЗ і, на думку багатьох фахівців, оптимальною.

Такий підхід до розробки ПЗ виконується шляхом паралельного виконання робіт, що включає в себе процеси безперервного аналізу отриманих в ході цього результатів і коригуванням попередніх етапів роботи . При цьому, ПЗ в кожній фазі проходить через цикл Плануй-Роби-Перевіряй-Дій (Plan-Do-Check-Act - PDCA).

Процес починається з простої реалізації невеликого набору вимог до ПЗ та ітеративного вдосконалення еволюціонуючих версій ПЗ, поки повна система не буде впроваджена і готова до розгортання . Тобто, згідно основної ідеї цього методу, система розроблюється повторюваними циклами (ітеративно) та невеликими «порціями» за один раз (інкрементно).

Життєвий цикл розробки ПЗ розбивається на міні-цикли (ітерації) замість єдиної послідовності етапів, кожен з яких включає свої власні етапи аналізу вимог, проектування, реалізації і завершується тестуванням, інтеграцією та створенням працюючої частини системи . Ітерація містить розробку окремого компонента(ів) системи, який далі додається до вже існуючого функціоналу. Таким чином, система поступово збільшується крок за кроком і може приймати «товарний вигляд» за 10-15 ітерацій (рис. 1.7).

Рисунок 1.7 - Ітеративна розробка ПЗ

Послідовні фази ітераційної моделі описано у таблиці 3

Таблиця 3 Фази ітераційної моделі

Назва фази	**Діяльність фази**
Планування	- складання специфікаційних документів; - загальної підготовки до майбутніх етапів циклу.
Збір вимог	- встановлення вимог до програмного чи апаратного забезпечення.
Аналіз та проектування	- визначити відповідну бізнес-логіку, моделі баз даних тощо, що буде треба на цьому етапі проекту; - встановлення будь-яких технічних вимог (мови, рівні даних, послуги тощо).
Впровадження	- кодування та впровадження усіх документів та специфікації в ітерацію проекту.
Тестування	- проходження ряду процедур тестування, щоб виявити та знайти будь-які потенційні помилки або проблеми, які виникли.
Еволюція	- ретельна оцінка розвитку проекту на цій стадії.
Розгортання	- розгортання системи у відповідному середовищі.

Перевагою даної моделі, перед попередніми розглянутими традиційними, є те, що вона не вимагає повного обсягу вимог для початку розробки . Вважається за достатнє – наявність вимог до базової частини функціоналу, а вже на наступних ітераціях, вимоги доповнюються, модифікуються і призводять до розширення функціоналу системи.

Від гнучких підходів ітераційну модель відрізняє те, що:

- вимоги до ПЗ розділені на кілька модулів, які можуть бути поступово розроблені та узгодженні. Тобто, спочатку розробляються основні функції, а все ПЗ розробляється шляхом додавання нових у такі версії. В той час, як у Agile модель передбачає процес доставки, при якому поставляється кожна поступово розроблена частина, яке не обов'язково являє собою цілий компонент ;
- у моделі ітеративного розвитку, як правило, немає фіксованого часу для завершення наступної ітерації, у Agile - дата завершення ітерації фіксована.

Ітеративний розвиток має деякі специфічні програми в індустрії ПЗ і найчастіше ця модель використовується за таких сценаріїв :

- визначені основні вимоги на початку проекту;
- використовується нова технологія, яку команда розробників вивчає під час роботи над проектом;
- деякий функціонал та цілі високого ризику, які можуть змінитися в майбутньому.

1.2.2 Каскадна модель

Каскадна (або водоспадна, лінійно-послідовна) модель є найстарішою, найпростішою і найвідомішою моделлю побудови багаторівневого процесу розробки ПЗ. Вона є дуже легкою і простою для розуміння, але, в той же час, занадто ідеалістичною і є не настільки практичною як раніше .

В умовах динамічних та швидко змінюючихся вимог, подібний строго структурований підхід до процесу може з переваги перетворитися в перешкоду на шляху успішного завершення розробки системи , Отже сьогодні водоспадна модель якщо і застосовується, то це переважно великими компаніями і лише для великих і складних проектів, які передбачають всеосяжний контроль ризиків.

Хоча на сьогоднішній день каскадна модель майже і не використовуються, але вона є дуже важливою, оскільки всі інші моделі життєвого циклу розробки ПЗ базуються саме на ній.

Класична модель ділить життєвий цикл на деякий набір фаз, кожну з яких можна розпочати лише після завершення попередньої, тобто вихід однієї фази є входом для наступної . І ось за такої ідеї, процес розвитку є «водоспадом» з послідовним потоком фаз, звідки і походить назва. В цьому і полягає основна відмінність каскадної методології від гнучких конкурентів: Agile, DSDM, Scrum, FDD і т.д.

Різні послідовні фази класичної моделі водоспаду показані нижче(рис 1.8)

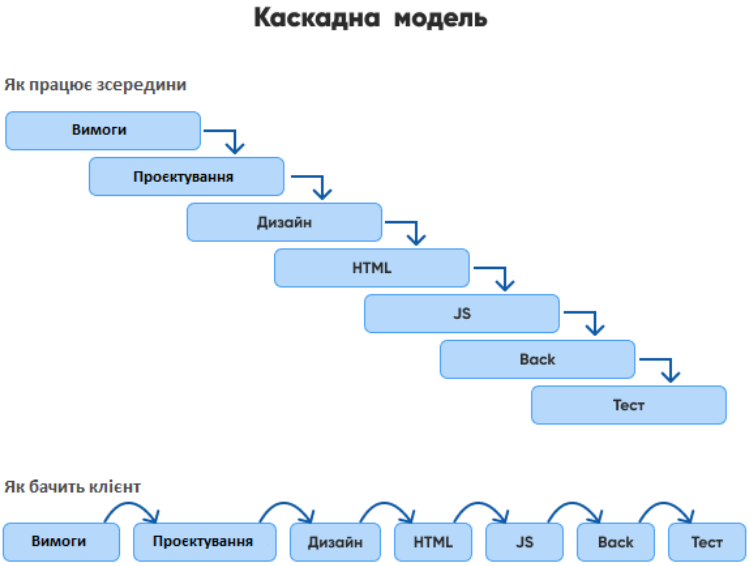


Рисунок 1.8 - Каскадна модель життєвого циклу ПЗ

Послідовні фази у моделі водоспаду розписано у таблиці 4

Таблиця 4 Послідовні фази моделі водоспаду

Назва фази	**Діяльність фази**
Виникнення ідеї та її обговорення	- розгляд ідеї, що виникла - техніко-економічне обґрунтування - визначення стратегій роботи
Аналіз та специфікація вимог	а) Збір та аналіз вимог: - збір вимог від замовника - аналіз зібраних вимог (усунення неповноти та невідповідностей). б) Специфікація вимог: - задокументовані в документі специфікації вимог до ПЗ;
Проектування	- перетворення вимог у структуру, яка підходить для реалізації на якійсь мові програмування. - вибір мови програмування (Java, PHP, .net), бази даних (Oracle, MySQL тощо), інших технічні деталей.
Кодування та модульне тестування	- дизайн ПЗ перекладається у вихідний код; - перевірка належної працездатності кожного модуля.
Інтеграція та системне тестування	А) Попередньо заплановані модулі додаються до частково інтегрованої системи та тестується отримана система. Б) Системне тестування: - альфа-тестування - це системне тестування, яке проводить команда розробників. - бета-тестування - це тестування системи, проведене доброзичливим набором клієнтів. - прийомне тестування - замовник проводить приймально-здавальне тестування, щоб визначити чи приймати поставлене ПЗ.
Розгортання системи	- розгортання системи у відповідному середовищі.

Технічне обслуговування	- коригувальне технічне обслуговування: виправлення помилок, які не були виявлені на етапі розробки продукту; - досконале технічне обслуговування: покращення функціональних можливостей системи на основі запиту замовника; - адаптивне обслуговування: перенесення ПЗ для роботи в новому середовищі.

Треба пам’ятати, що кожне розроблене ПЗ відрізняється і вимагає відповідного і не схожого на інші підходу до розробки ПЗ, який слід застосовувати, спираючись на внутрішні та зовнішні фактори. Деякі ситуації, коли використання моделі водоспаду є найбільш доцільним :

- вимоги, зареєстровані в документі, є незмінними та чіткими;
- нетривалий проект;
- технології та інструменти не є динамічними та є стабільними;
- для підтримки даного продукту достатньо ресурсів;
- замовник бере участь лише на початкових етапах, а згодом – – приймає готовий продукт;
- основний пріоритет - якість, навіть на шкоду часу;
- допускається можливість виконання проекту на аутсорс.

1.2.3 V-модель

V-модель є «покращеною» версією розглянутої каскадної моделі розробки ПЗ, де виконання процесів відбувається послідовно, у V подібній формі. Вона також відома як модель валідації та верифікації:

- *Верифікація* - це модель перевірки документів, архітектури, дизайну, коду тощо, яка включає техніку статичного аналізу (огляд) , виконану без виконання коду, та відповідає на питання «Чи робимо ми продукт правильно?».
- *Валідація* - це процес оцінки кінцевого продукту, необхідно перевірити, чи відповідає ПЗ очікуванням і вимогам клієнта, яка включає техніку динамічного аналізу та відповідає на питання «Чи робимо ми продукт правильним?».
-

Отже, V-модель містить фази верифікації з одного боку та фази валідації з іншого боку, які об’єднуються кодуванням у V-подібній формі, саме звідси і походить назва моделі.

Кожен етап моделі супроводжується контролем поточного прогресу, що дає можливість оцінити наскільки можливо та доцільно переходити на наступний рівень проекту згідно із моделлю . При цьому, ще зі стадії написання вимог починається процес тестування та продовжується для кожного наступного етапу відповідно до того, які очікувані результати та критерії входу/виходу прописано у сформованому тест-плані.

Як видно із пояснення, V-модель дотримується проходження через процес тестування на кожному етапу проектування і розробки системи відповідно до визначеного рівня. Тут процес розробки представлений низхідною послідовністю в лівій частині умовної букви V, а стадії тестування - на її правому ребрі.

Відповідність етапів розробки та тестування показано горизонтальними лініями(рис 1.9)

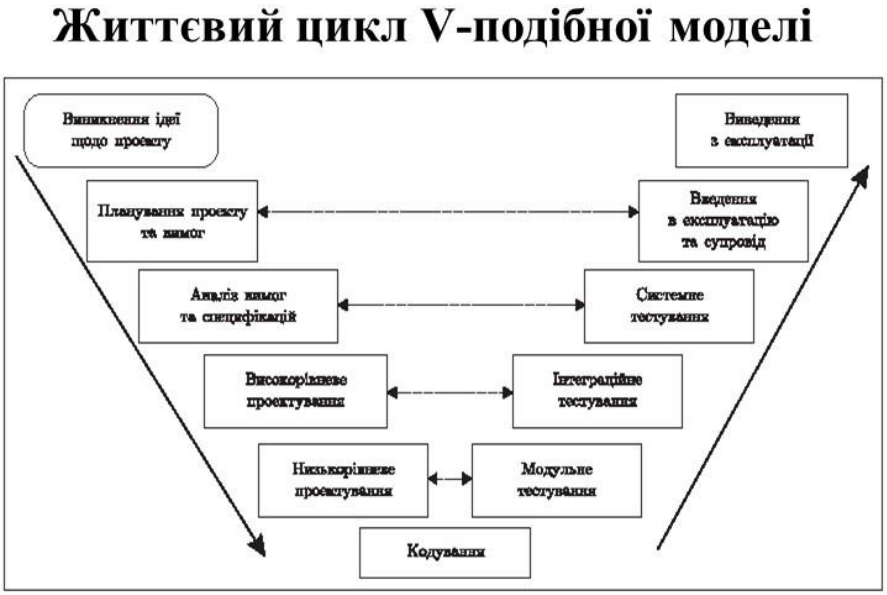


Рисунок 1.9 - V-модель життєвого циклу розробки ПЗ

Послідовні фази у моделі водоспаду описано таблиці 5

Таблиця 5 Фази моделі водоспаду

Назва фази	**Діяльність фази**
Аналіз бізнес-вимог	- детальний зв’язок із замовником з метою визначення очікувань та точних вимог; - планування приймально-здавальних випробувань проекту.
Дизайн системи	- розробка структури системи (повна апаратурна та комунікаційна установка); - розробка плану тестування;
Архітектурний дизайн	- розробка архітектурних специфікацій (передача даних та зв’язок між модулями та зовнішніми

	системами); - розробка та документація інтеграційних тестів.
Проектування модулів	- визначення детального внутрішнього проекту для системних модулів; - розробка та документування модульних тестів;
Кодування	- фактичне кодування модулів
Тестування	- модульне тестування: тестування на рівні коду; - інтеграційне тестування: перевірка співіснування та взаємодії внутрішніх модулів; - системне тестування: перевірка всієї функціональності системи та зв'язку системи із зовнішніми системами; - прийомне тестування: перевірка продукту в середовищі користувача (виявлення проблем сумісності з іншими системами користувача, тестування навантаження).

V-модель виділяє такі принципи розробки:

- ***Від великого до малого.*** Відповідно до цього принципу V-модель передбачає ієрархічне тестування, за якого вимоги спочатку визначаються командою проекту, потім в ході проектування високого та детального рівнів. Завдяки цьому, вимоги стають дедалі більш досконалими та деталізованими .
- ***Цілісність даних/процесів.*** За дотримання цього принципу елементи процесу визначаються з дотриманням кожної вимоги, що дає змогу досягти згуртованості всіх даних і процесів .

- ***Масштабованість.*** Концепція моделі передбачає гнучкість для розробки будь-якого IT-проекту (різний розмір, складність або тривалість).
- ***Перехресне посилання.*** Пряма кореляція між вимогами та тестами.
- ***Матеріальна документація.*** Кожен проект повинен створити документацію, яка використовується для ведення проекту.

Доцільність застосування V-моделі орієнтовно майже така ж, як у каскадної моделі, оскільки обидві моделі мають послідовний тип . Можна виділити такі сприятливі умови використання даної моделі:

- чіткі й задокументовані вимоги (Отже що повертатися назад і вносити зміни дорого);
- стабільне та усталене визначення продукту;
- доцільність для малих та середніх проектів;
- достатня кількість технічних ресурсів із необхідною технічною експертизою;
- висока довіра замовника (оскільки прототипи не виробляються й існує дуже високий ризик незадоволення очікувань споживачів).

1.2.4 Спіральна модель

Спіральна модель – модель розробки ПЗ, яка забезпечує підтримку управління ризиками, на схематичному відображенні дана модель відображається у вигляді спіралі (фази) із безліччю петель, кількість яких може варіюватися в залежності від проекту :

- радіус спіралі - витрати (вартість) проекту на даний момент;
- кутова розмірність - прогрес, досягнутий на даний момент на поточній фазі.

Отже спіральна модель – модель, яка є подібною до «поступового» розвитку системи, який було приведено вище, але з більшим акцентом на аналізі ризиків . Нижче на схемі(рис 1.10) показані різні фази спіральної моделі та розділення фаз моделі на квадранти

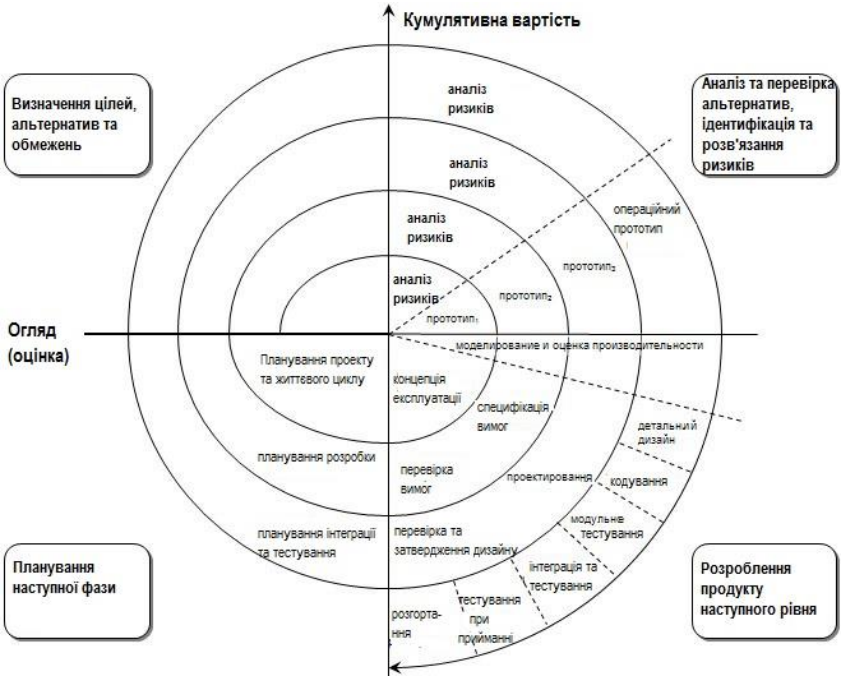


Рисунок 1.10 - Спіральна модель життєвого циклу ПЗ

Квадранти фаз спіральної моделі представлені у таблиці 6

Таблиця 6 Фази спіральної моделі

Назва квадранту	**Функції квадранту**
Визначення цілей та альтернативних рішень	- збір вимог споживачів; - визначення, розробка та аналіз цілей; - визначення можливих альтернативних рішень.
Визначення та вирішення ризиків	- оцінка можливих рішень; - ідентифікація ризиків рішення; - розробка стратегії для ризиків; - розробка прототипу найкращого рішення.

Розробка версії продукту	- розробка та тестування визначеного функціоналу; - доступна версія ПЗ.
Огляд та планування наступного етапу	- оцінка розробленої версії ПЗ замовником; - планування наступного етапу.

Ризик, в даному випадку, розглядається як несприятлива ситуація і можливість її впливу на успішність завершення проекту ПЗ. І спіральна модель передбачає управління цими невідомими та несприятливими ситуаціями після початку проекту шляхом розробки їхнього «прототипу» на кожному етапі розробки ПЗ . Існує ще таке поняття, як модель прототипування, що підтримує управління ризиками, але в цьому випадку вони повинні бути повністю визначені перед початком роботи. Проте, мінусом її є те, що, на жаль, в реальному житті ризик може виникнути вже після початку роботи, і в цьому випадку ми не можемо використовувати модель прототипування. Саме Отже на кожному етапі спіральної моделі визначаються ризики та вирішуються шляхом створення їхніх прототипів.

Таким чином, ця модель набагато гнучкіша порівняно з іншими наведеними. Спіральну модель ще іноді носить іншу назву - мета-модель, внаслідок того, що вона включає у себе всі інші розглянуті вище моделі. Так, наприклад, спіраль з однією петлею насправді представляє ітеративну модель водоспаду, крім того, спіральну модель можна розглядати як підтримку ітераційної моделі, оскільки ітерації вздовж спіралі можна розглядати як еволюційні рівні, за допомогою яких будується повна система .

Отже, спіральна модель поєднує ідею ітеративного розвитку із систематичними, контрольованими аспектами моделі водоспаду, Отже що є комбінацією ітеративної моделі процесу розвитку та моделі водоспаду з дуже високим акцентом на аналізі ризиків . Завдяки цьому, існує можливість

здійснювати поступову доставку продукту або поступове вдосконалення через кожну ітерацію навколо спіралі.

Спіральна модель є широко застосованою в індустрії ПЗ й такі тези відображають типове використання спіральної моделі :

- обмеженість бюджету;
- важливість оцінки ризику;
- складні та динамічні вимоги;
- вимога поетапного випуску ПЗ;
- великий проект.

1.3 Висновки до розділу

Моделі життєвого циклу ПЗ за велику кількість років були удосконалені та перероблені, розроблено нові моделі ЖЦ. Все це робиться з метою, покриття попиту, а також очікувань розвитку. У світі не існує такої моделі, яка б сто відсотково підійшла до усіх проектів. Навіть багатоцільовий та універсальний Agile неможливо широко використовувати, внаслідок небажання деяких клієнтів й замовників масштабувати бюджет згідно із мінливих вимог.

Традиційні підходи використовують планування як механізм управління, у той час як моделі Agile використовують зворотний зв'язок від користувачів як головний механізм управління, Отже цей підхід можна назвати найбільш орієнтованим на людей та кінцевого користувача, ніж традиційні методи. В порівнянні із плановими підходами, гнучкі доставляють робочу версію ПЗ набагато раніше, час «ітерацій» тестування порівняно короткий, внаслідок того, що здійснюється це паралельно з кодуванням.

Отже, існує безліч моделей розробки ПЗ, але вибір того чи іншого варіанту залежить від особливостей і вимог проекту, можливостей оплати. Звісно, частково методології перетинаються і схожі один на одного, але кожна має своїх шанувальників залежно від потреб та особливостей.

2 УПРАВЛІННЯ ВАРТІСТЮ І ЯКІСТЮ ПРОЦЕСІВ ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

2.1 Управління часом проекта

Управління часом - це одна з найважливіших підсистем проекту, поряд з витратами та результатами входить до «трикутника» проекту та визначає обмеження проекту за часом. управління часом складає всіх етапах життєвого циклу проекту, реалізуючись у різних функціях проект-менеджмента. На етапі розробки проекту - це планування часу проекту, на етапі реалізації - контроль виконання мережного графіка та внесення змін у ході здійснення проекту.

Головним завданням управління часом на етапі планування є розробка такого розкладу робіт, при якому цільова функція при дотриманні всіх умов досягала б екстремального значення. Тобто, головне завдання календарного планування інтегрує у собі досягнення трьох умов:

- Мінімізація тривалості проекту в умовах обмеженості ресурсів;

- Мінімізація вартості проекту;

- Рівномірний розподіл ресурсів.

Результатом виконання головного завдання планування часу є - обґрунтований календарний план.

Календарний план - це проектно-технологічні документи, що встановлюють повний перелік робіт проекту, їх послідовність, взаємозв'язок, терміни виконання, тривалість, виконавців та ресурси, необхідні для виконання робіт.

Створення календарного плану передбачає низку попередніх дій:

1) визначення тривалості робіт;
2) встановлення взаємозв'язку між роботами;

3) визначення часу доступності всіх видів ресурсів.

Процес визначення тривалості робіт може здійснюватися по-різному, зокрема методом *Delphi*, з використанням баз даних, за допомогою внутрішніх та зовнішніх консультантів, існуючих стандартів і т. д. при застосуванні методу *Delphi* експерти письмово, незалежно одне від одного оцінюють ситуацію. після цього кожен експерт знайомиться з оцінками колег та коригує свою оцінку. Процедура повторюється до того часу, поки оцінки зблизяться до прийнятного інтервалу часу.

Метод не застосовується при виробництві унікальних інноваційних робіт. Кількісний метод враховує обсяг робіт та вироблення - доцільність праці.

У цьому методі передбачається можливим врахувати основні чинники тривалості роботи: трудомісткість, кількість виконавців, чистий час затримки.

Трудомісткість роботи - це час, необхідний одній людині на виконання даної роботи. Вимірюється в людино-годинах. Неважко підрахувати, як тривалість виконання залежить кількості працівників.

На тривалість виконання також впливає ефективність використання робочого часу, тут слід враховувати певні закономірності.

Занадто оптимістичний розрахунок на використання всього робочого часу для безпосереднього виконання проекту загрожує нереалістичною тривалістю завдань, яка може обернутися зривом термінів проекту, насамперед Отже, що час виконання завдання завжди подовжується за рахунок виконання інших службових обов'язків (наради, обговорення тощо) , і навіть за рахунок непередбачених особистих причин, коли працівник буває недоступний (хвороба, незаплановані перерви, форс-мажор). за дослідженнями американських авторів реальна тривалість робочого дня протягом стандартного робочого дня (у годинах) виглядає так (табл 7).

Таблиця 7 Тривалість робочого часу в залежності від ефективності використання годин

трудомісткість	100-процентна ефективність, 100-відсоткова доступність	75-процентна ефективність, 100-відсоткова доступність	75-процентна ефективність, 75-відсоткова доступність
1 чол.-день	8	6	4,5
1 чол.-тиждень	40	30	22,5
1 чол.-місяць	173	130	98
1 чол.-рік	2080	1560	1170

Чистий час затримки пов'язаний із тривалістю, яка не залежить від трудомісткості робіт, і стосується таких ситуацій, які вимагають роботи з документами: затвердження, отримання дозволів, сертифікатів тощо. трудомісткість робіт може становити кілька годин, а вся процедура триватиме п'ять днів; таким чином, тривалість виконання роботи становитиме п'ять днів. наочне уявлення про тривалість робіт дає діаграма ганта(рис. 2.1).

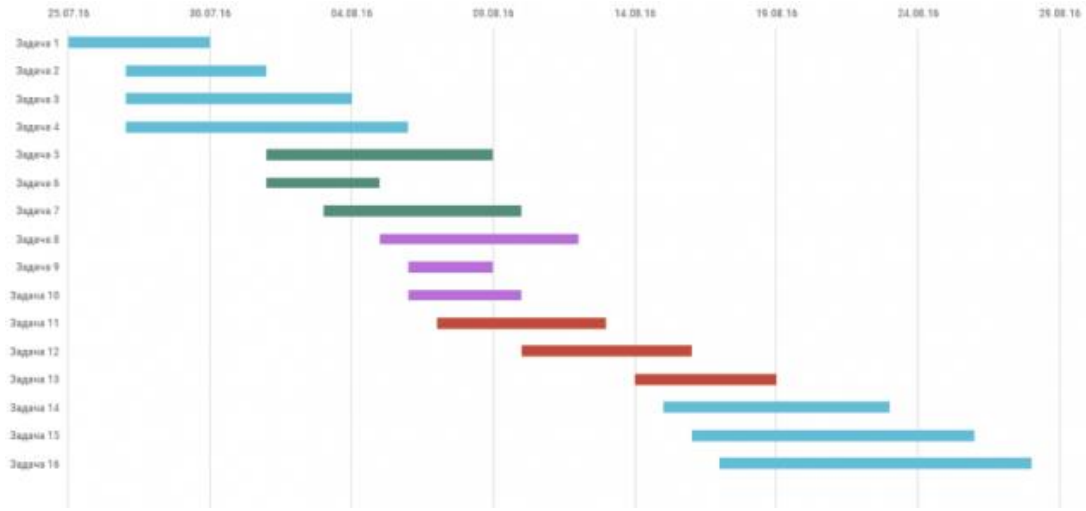


Рисунок 2.1 - Діаграма Ганта

Послідовність та взаємозв'язки робіт відображаються на мережевому графіку. Існує два види мережевих графіків: традиційний(рис 2.2) та графік PERT(рис 2.3). Традиційний графік, побудований за принципом події-роботи, графік PERT (Program Evaluation and Review Technique - метод оцінки та перегляду плану) - за типом роботи-зв'язку.

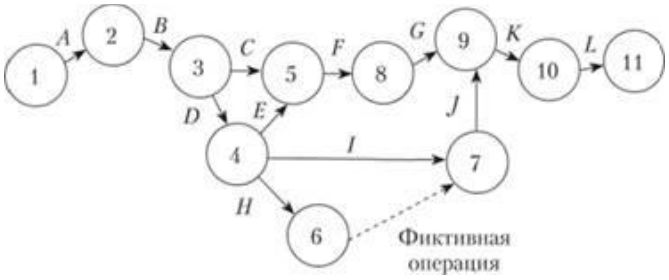


Рисунок 2.2 - Традиційний мережевий графік

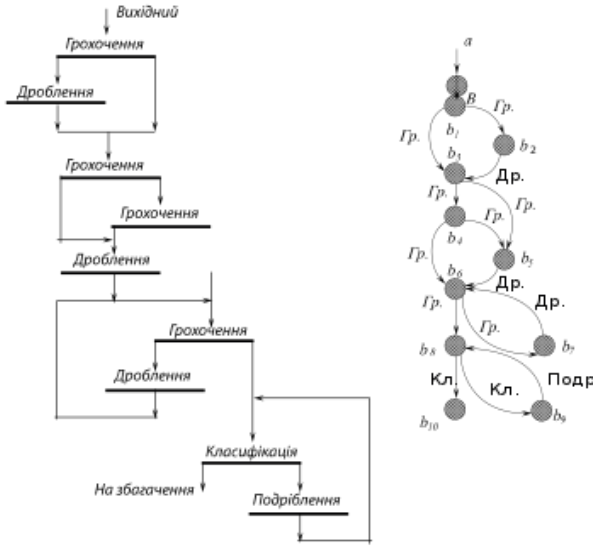


Рис. - Схема технологічного процесу і граф.
b_i - стан матеріалу; ***Гр.*** - грохочення; ***Др.*** - дроблення;
Кл. - класифікація; ***Подр.*** - подрібнення.

Рисунок 2.3 - Мережевий графік PERT

У сучасній практиці найчастіше використовується саме мережевий графік «роботи-зв'язку», він набагато зручніший, оскільки може відображати і роботи, і події, і, крім того, саме за цим принципом працює і комп'ютерна програма Microsoft Project , найпопулярніша у повсякденній практиці проект-менеджменту.

основні правила мережевого графіка:

1) після завершення попередньої роботи можна приступати до виконання наступної, до якої йдуть стрілки;

2) розпочати роботу можна, тільки завершивши всі попередні роботи, від яких стрілки ведуть до шуканої роботи.

Для розуміння сенсу мережного планування необхідно також дати визначення ключових понять мережного графіка.

Критичний шлях проекту - це послідовність робіт проекту, яка вимагає найбільше часу для завершення, тобто це найтриваліший ланцюжок робіт. всі роботи, що лежать на цьому шляху, називаються «критичними завданнями», і незаплановане подовження будь-якої з них призведе до подовження всього проекту. очевидно, що саме довжина критичного шляху визначатиме термін виконання всього проекту . поняття критичного шляху дозволяє проводити планування як від дати початку проекту, і від фіксованої дати закінчення, що дуже зручно для проект-менеджера; тоді першому випадку необхідно визначити дату закінчення, тоді як у другому — початку робіт.

Критичний шлях визначається обчисленням раннього та пізнього Початоку (*Early Start , Late Start*) та фінішу (*Early Finish , Late Finish*) для кожної з робіт.

Некритичний шлях проекту - послідовність робіт, котру можна виконати з деякою затримкою, яка не призводить до збільшення тривалості проекту. Це відбувається через те, що некритичний шлях по визначенню коротше критичного і містить певний резерв часу, завдяки якому будь-яке завдання, що лежить на некритичному шляху, має певний тимчасовий люфт і може пересуватися по осі часу. таким чином, резерв часу — максимальний час, на який можна зрушувати завдання, яке лежить на некритичному шляху, без збільшення термінів проекту.

Завдяки пересуванню некритичних завдань по осі часу виникає можливість визначити точні терміни раннього початку - закінчення і найпізнішого початку - закінчення робіт.

Для визначення довжини критичного шляху та встановлення термінів раннього початку - закінчення проекту проводиться прямий аналіз мережного графіка. Для встановлення пізніх термінів початку - закінчення та відповідно величини резервів часу - зворотний аналіз.

Зрозуміло, на практиці використовуються і більш складні залежності та зв'язку. конкретний графік залежить від багатьох причин: від складності послідовностей та зв'язків, від завдань, що стоять перед проект-менеджером, від типу програмного забезпечення проекту тощо.

Підсумковим документом планування часу, є календарний план(табл. 9). стандартний календарний план повинен містити конкретні терміни, відомості про резерви часу та прізвища відповідальних членів команди.

Сучасне програмне забезпечення дозволяє поєднати всі ці відомості на діаграмі ганта , вказавши ще й потрібні види ресурсів.

Залежно від ситуації форма представлення календарного плану то, можливо інший. Наприклад, це можливо графік ключових подій «план по віхам».

календарний план може бути представлений у традиційній формі як звична таблиця(табл. 8) із зазначенням дат подій та відповідальних за виконання.

Таблиця 8 Зазначенням дат подій та відповідальних за виконання

подія	березень	квітень	травень	червень	липень	серпень
контракт укладений	△▽					
оформлена специфікація			△▽			
розроблений дизайн системм				△		
Підключеня протестованої системи					△	
проект завершений						△

позначення: △ - Запланована дата; △ - Фактична дата.

Таблиця 9 Календарний план

№ п/п	назва завдань	початок	закінчення	резерви	відповідальний
1	підготовка документів	03.03.08	10.03.08	критичне завдання	іванова л. п.

Можна представляти календарний план у формі, запропонованої у Microsoft project , вона дуже зручна для повсякденної управлінської діяльності.

При розробці розкладу проекту може виникнути ситуація ресурсного конфлікту. ресурсні конфлікти - це невідповідність між межею споживання ресурсу та потребою в даному ресурсі для виконання роботи.

Методи вирішення конфліктів:

- Стиснення;

- Розтяг;

- Нормалізація;

Стиснення розкладу робіт призводить до скорочення термінів проекта , але збільшує ризики.

Методи стиснення:

- *Crashing* - залучення додаткових ресурсів для прискорення виконання робіт, що знаходяться на критичному шляху (купівля додаткових ресурсів; робота у позаурочний час; перерозподіл ресурсів із завдань, що не знаходяться на критичному шляху);

- *Fast Tracking* - паралельне виконання фаз або робіт проекту, які у звичайній практиці виконуються послідовно.

саме паралельне виконання робіт проекту дозволяє зі-

кратити критичний шлях та, відповідно, терміни проекту. при цьому способі стиснення виникають некритичні шляхи проекту.

2.2 Управління вартістю проекта

Управління вартістю проекта відбувається на всіх етапах життевого циклу проекту і включає такі процеси, що забезпечують виконання проекту в рамках затвердженого бюджету:

- Вартісна оцінка;

- Розробка кошторису та бюджету проекту;

- Контроль вартості (*Cost Control*).

Таким чином, метою управління вартістю є розробка процедур та методів, що допомагають планувати витрати та своєчасно їх контролювати за допомогою різних методів.

Процеси управління вартістю реалізуються по-різному на різних етапах життєвого циклу і сама вартість проекту розподіляється нерівномірно протягом життєвого циклу(рис. 2.4). Основна частина вартості витрачається на реалізацію, здійснення проекту, але слід пам'ятати, що основні рішення, що зумовлюють показники вартості проекту, приймаються на передінвестиційну фазу. Звідси виникає не лише найважливіше значення цієї фази, а й облік можливості управління вартістю — вона зменшується пропорційно до закінчення.

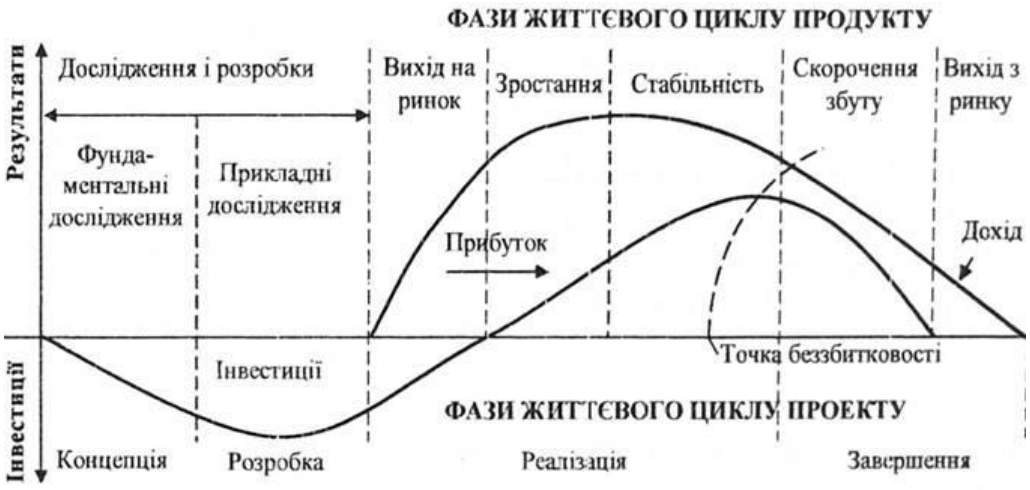


Рисунок 2.4 - Розподіл коштів за фазами життєвого циклу

Вартісна оцінка (*Cost Estimating*) - визначення вартості ресурсів, необхідних для виконання операцій (завдання цільової структури) проекту:

- Обладнання (купівлі або оренди);

- Пристроїв (пристроїв та виробничих потужностей);

- Робочої праці (штатного персоналу та контрактників);

- Витратних матеріалів (канцелярських товарів та ін);

- Сировини та матеріалів;

- Навчання, семінарів, конференцій;

- Субконтрактів;

- Транспортних витрат.

Існують різні методи та види оцінки вартості проекту, представлені види та цілі оцінки залежно від етапів проекту.

Методи та засоби оцінки вартості ресурсів представлені на таблиці 10:

- *Оцінка за аналогами* - за аналогією з минулими схожими проектами або роботами;

Таблиця 10 Методи оцінки вартості проекту

Стадії здійснення проекту	Види оцінок	Ціль оцінок
концепція проекту	Попередня оцінка життєздатності проекту	Оцінка реалізованості проекту
обґрунтування інвестицій	Укрупнений розрахунок вартості - попередній кошторис	Зіставлення запланованих витрат із бюджетом організації
розробка робочої документації	Остаточна кошторисна документація	Основа для розрахунків та управління вартістю проекту
реалізація проекту	Фактична (за вже виконаними роботами).	Оцінка вартості вже виконаних работ.

	Прогнозна (за майбутніми роботами)	Оцінка вартості майбутніх реалізації робіт
здача в експлуатацію	Фактична. Прогнозна	
завершення проекту	Фактична	Повна оцінка вартості проекту

- *Визначення ставок вартості ресурсів* - оцінка за параметрами проекту (вартість 1 години роботи + вартість одиниці матеріалу);

- *Оцінка «знизу - вгору»* - Оцінка вартості окремих робіт, потім пакетів робіт і т. д. (від нижнього до верхніх рівнів WBS).

Отже, за для визначення вартості проекту потрібна наступна інформація:

- Час виконання робіт;
- Вартість складових проект ресурсів;
- Вартість робіт.

Ціна проекту визначається сукупністю цін ресурсів проекту. Основним документом, з допомогою якого здійснюється управління вартістю проекту, є *бюджет проекту* **.**

Бюджет — це директивний документ, що є реєстром планованих *витрат і доходів* з розподілом за статтями на відповідний період часу. бюджет - документ, що визначає ресурсні обмеження проекту.

бюджет може бути сформований у межах традиційного бухгалтерського обліку. в залежності від стадії життєвого циклу проекту бюджети можуть бути:

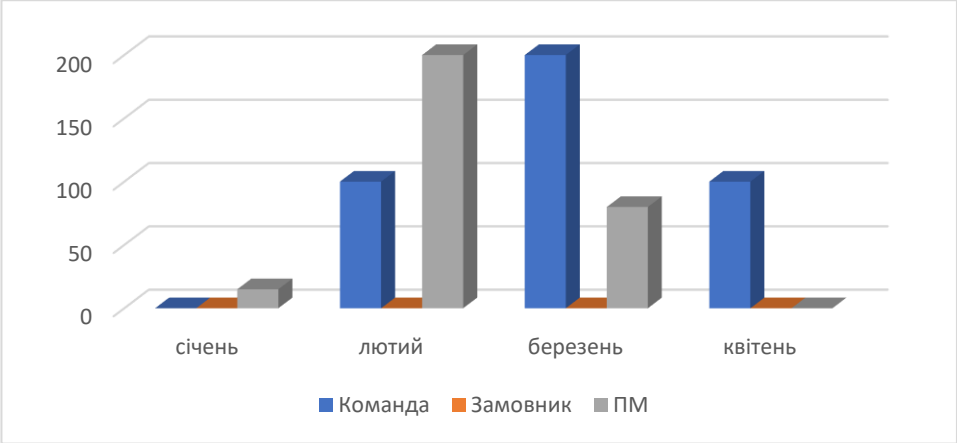
- Попередні (оціночні);
- Затверджені (офіційні);
- Поточні (кориговані);

- Фактичні.

сутність бюджетування - це планування вартості проекту, тобто певного плану витрат: коли, скільки і за що будуть виплачені та отримані гроші. витратна складова бюджету називається кошторисом. Методи представлення кошторису витрат може бути різні і залежати від цілей документації, сформованих традицій та побажань замовника. кошторис може бути представлений у вигляді календарних планів-графіків(табл. 11), стовпчастих діаграм, стовпчастих діаграм кумулятивних витрат, лінійних діаграм розподілених за часом кумулятивних витрат(діаг. 1), кругових діаграм(діаг. 2), що відображають структуру витрат на проект.

Таблиця 11 Календарний план-графік витрат

№ п/п	роботи	січень	Лютий	Березень	квітень	Травень
1	прийняти справи	15$				
2	Плануватття та вводення даних		300$			
3	звірити базу			550$		
4	сформувати пакет				280$	
5	оформити архів					100$



Діаграмма 1 - Стовпчаста діаграма кумулятивних витрат

Діаграмма 2 - Кругова діаграма витрат, %

Після прийняття, погодження та затвердження бюджет та кошторис стають зразком, з яким порівнюють фактичний результат, та основним документом проекту.

Контроль вартості проекту є частиною управління змінами і містить в собі пошук причин, що викликають як позитивні, так і негативні відхилення. Наприклад, невчасне реагування на відхилення вартості може призвести до виникнення проблем з розкладом або якістю, до появи неприйнятного збільшення ризику на подальших етапах проекту.

контроль вартості включає такі процедури:

- Встановлення фактичної вартості проекту;

- Порівняння фактичної вартості з планової;

Прогноз майбутньої загальної вартості проекту. існує два основних методи контролю вартості:

- традиційний та метод освоєного обсягу.

Традиційний метод дає хороший результат для визначення стану справ після закінчення проекту для визначення розбіжності фактичної та планової вартості проекту. при застосуванні традиційного методу вводяться такі показники:

Planned Value (PV) – плановий обсяг, планова вартість запланованих робіт.

планові бюджетні витрати: BCWS (Budgeted cost of work

scheduled).

ACWP (Actual cost of work performed) - фактична вартість виконаних робіт, сума коштів, фактично витрачена на виконання робіт до фіксованої дати, яка не залежить від бюджетних планових показників.

Зіставлення останніх двох величин дає нам розбіжність у вартості, розбіжності за витратами (COST VARIANCE) і дозволяє визначити перевитрату або економію коштів:

CV = ACWP - BCWS.

Другий метод зручний саме для проекту, Отже що враховує наявність графіка робіт і дозволяє встановити не тільки відхилення витрат, але ще й відхилення від графіка робіт.

Одна з найпоширеніших технологій вартісного аналізу - *аналіз досягнутого обсягу* (Earned Value Analysis). аналіз досягнутого обсягу – *інтегрований аналіз* як виконання календарного плану проекту, і бюджету за вартісними показниками. для цього до двох визначених у традиційному методі показників додається третій – бюджетна вартість фактично виконаних робіт Earned Value (EV) або CWP (budgeted cost of work performed) - Освоєний обсяг, планова вартість виконаних робіт. Освоєний обсяг залежить від фактично виконаних вироблених витрат за роботі.

Відхилення за витратами є величиною, різниці фактичної вартості виконаних робіт (ACWp) і планової вартості фактично виконаних робіт (в CWP). Ця залежність відбито на графіку(рис 2.5).

CV = ACWp - BCWS.

Відхилення від графіка визначається різницею між планової вартістю робіт за графіком BCWS та плановою вартістю фактично виконаних робіт BCWP.

T = BCWS - BCWp .

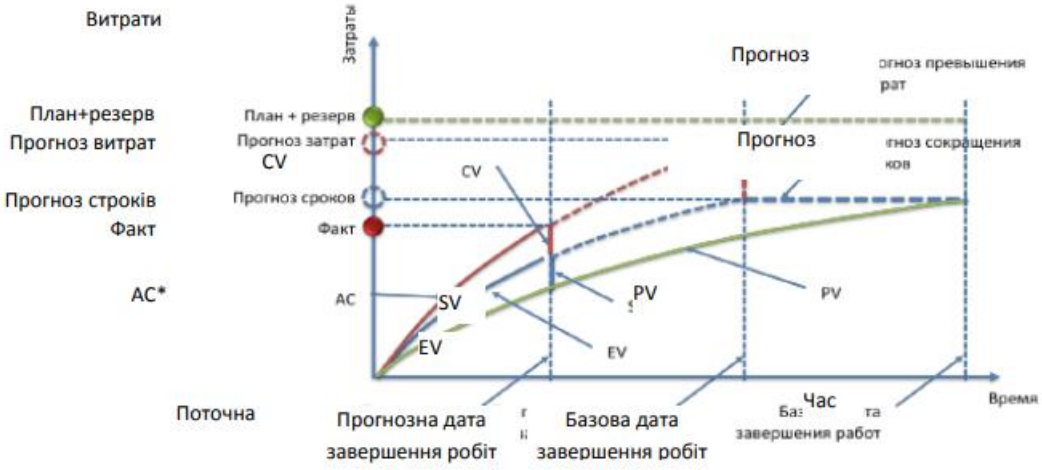


Рисунок 2.5 - Аналіз освоєного обсягу та розбіжностей

*EV - освоєний обсяг, AC – фактична вартість, PV – плановий обсяг,CV - відхилення за витратами, SV -відхилення за розкладом

Одне із завдань контролю вартості - встановлення прогнозної оцінки вартості проекту на підставі інформації про витрати проекту на поточний момент часу. оцінку кінцевої вартості проекту можна провести традиційним методом та методом освоєного обсягу.

CPI (Cost Performance Index) - індекс виконання бюджету, він визначається за формулами

CP = EV - AC,

CPI = BCWP - ACWP.

CPIc (Cumulative CPI) - накопичувальний індекс виконання бюджету, він визначається за формулою:

CPIc = EVc ÷ Acc .

SPI (Schedule Performance Index) — індекс виконання календарного плану, що визначається за формулами

SPI = EV ÷ PV, SPI = BCWP ÷ BCWS.

Таким чином, можна зробити прогноз про вартість(табл 12) як усього проекту в цілому, так і його частини, що залишилася, на момент завершення проекту.

Таблиця 12 Методи прогнозної оцінки вартості

ETC	EAC	примітка
нові оцінки робіт, що залишилися	Acc + нові оцінки робіт, що залишилися	Метод на основі нових оцінок - найбільш точний метод
BAC - EVc	Acc + BAC - EVc	Метод на основі нетипових змін (якщо вироблені витрати відрізнятимуться від майбутніх)

(BAC - EVc) / CPIc	Acc + ((BAC – EVc) / CPCc)	Метод на основі типових відхилень (якщо вироблені витрати будуть схожі на майбутні)

У практиці використовуються такі умовні скорочення:

- BAC (Budget Completion) - Планова вартість всього проекту;

- ETC (Estimate to Completion) - оцінка вартості частини проекту, що залишилася;

- EAC (Estimate at Completion) - Оцінка вартості проекту при завершенні. проілюструємо теоретичний матеріал на рішенні кон-

конкретного завдання. приклад у проекті будівництва будинку планова продуктивність - 1 поверх за 3 тижні при плановій вартості 1-го поверху 123 250 дол.

знайти відхилення за термінами (SV) та вартістю (CV), якщо до кінця 3-го місяця (у місяці 4 тижні) було закінчено 5 поверхів, а вартість виконаних робіт склала 630 750 дол **.**

PV = 3 · 4/3 · 123 250 = 493 000 дол.

TV = 5 · 123250 = 616250 дол.

AC = 630750 дол.

CV = EV - AC = -14 500 дол - перевитрата коштів;

SV = EV - PV = 123 250 дол. - Випередження за термінами.

3 ОГЛЯД СУЧАСНИХ КОНЦЕПЦІЙ УПРАВЛІННЯ ІТ ПРОЕКТАМИ.

3.1 Trello проти Asana: у чому різниця?

Якщо ви абсолютно новачок у всьому цьому мраку управління проектами, ви можете просто шукати швидке пояснення різниці між Asana і Trello.

Trello - це безкоштовна дошка канбану, яка додає тонну додаткових функцій, як тільки ви платите за це. Asana також пропонує тонну безкоштовних функцій, але додає ще більше можливостей, як тільки ви покладете свої гроші.

3.1.1Особливості

Почнемо з функцій. Через те, як Асана і Трелло побудували свої плани, буде багато збігів у наступних порівняннях, які фокусуються на ціноутворенні. Це означає, що в деяких конкретних випадках вам може знадобитися підбити висновки з порівнянь відповідно до власних потреб, перш ніж приймати будь-які рішення про покупку.

Що стосується того, що краще, коли справа доходить до функцій, коротка відповідь полягає в тому, що це Асана. Замість того, щоб пропонувати різні здібності з коробки, Trello покладається на так звані бонуси, щоб додати до своєї функціональності. Це інтеграції, як сторонні, так і розроблені самим Trello, і вони працюють унікальним чином.

3.1.2Trello проти Асани

Для початку, безкоштовна версія Asana має набагато більше для неї з точки зору функцій, ніж у Trello. Хоча його канбанова дошка не така хороша, вона набагато гнучкіша, ніж її конкурент, просто Отже, що вона пропонує більше.

Крім дошки, ви отримуєте перегляд списку, календар, необмежену кількість проектів (Trello обмежує безкоштовних користувачів до 10) і необмежене зберігання файлів.

Крім того, Asana і Trello управляли тим, чого не мають більшість хмарних сховищ: фактичне необмежене хмарне сховище. Розмір файлу обмежений 100 МБ для Asana і 10 МБ для Trello, але це все ще хороша угода.

Якщо ви хочете більше, ніж ці функції, вам доведеться поні, і досить серйозно, теж. Перший платний рівень Asana, Premium, становить 11 доларів США на користувача на рік, якщо виставляють рахунок щорічно, тоді як наступний, Бізнес, становить 25 доларів США.

Однак ви отримуєте те, за що платите, і ви отримуєте багато з Asana, від перегляду часової шкали до планувальників робочого навантаження або кількох подань дошки.

Asana має ще одну перевагу, а саме, що всі її функції розроблені і інтегровані Asana. Trello не завжди сам розробляє ці інтеграції, а це означає, що вони не завжди працюють так гладко.

Додайте до цього той факт, що деякі бонуси вимагають іншої оплати за їх використання, і у вас є деякі дивні коктейлі та дивні збіги, а також деякі додаткові рахунки для оплати.

Таким чином, я вважаю, що Asana є переможцем, коли справа доходить до функцій. Це просто працює краще, незалежно від того, чи керуєте ви кількома проектами або просто невеликою командою.

3.1.3Ціноутворення

Як я вже казав на початку останнього розділу, що існує деяке дублювання між ціновими та функціональними критеріями при порівнянні Trello проти Asana. Так само, як і в останньому раунді

Коли ви додаєте все це, Asana несе в собі трохи більше цінності, ніж Trello.

Цінність є досить суб'єктивним критерієм, особливо при порівнянні безкоштовних продуктів, але мені це подобається більше, ніж просто порівнювати программи в бекенді. Хоча Asana трохи дорожче, ніж Trello, вона просто пропонує більше, як я обговорювали в розділі функцій. Перш ніж я перейду до будь-якого наступного розділу, давайте подивимося на деякі цифри(рис 3.1, 3.2).

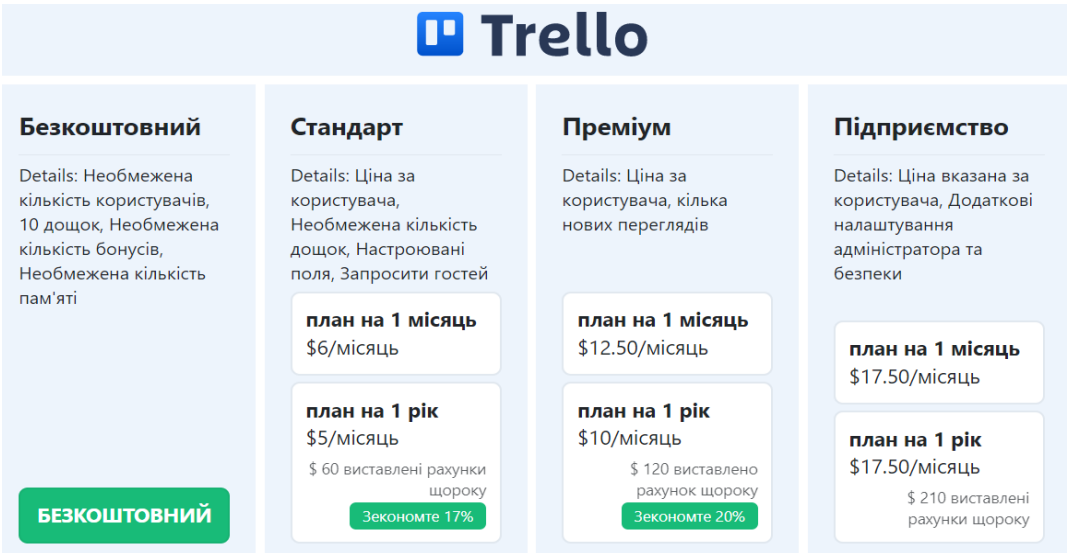


Рисунок 3.1 - Ціни Trello

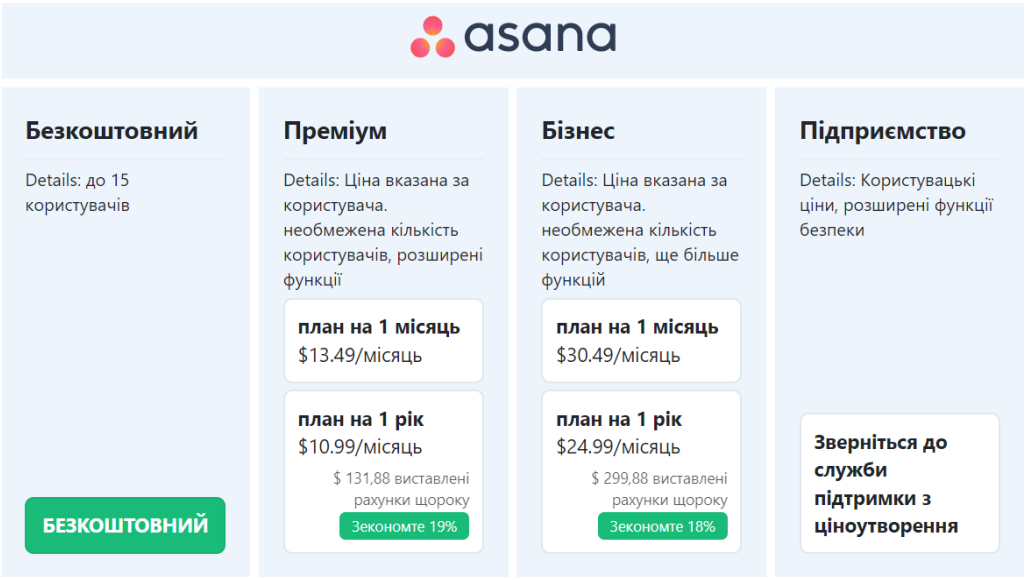


Рисунок 3.2 - Ціни на Asana

Всі ціни вказані за користувача на місяць і можуть бути оплачені кредитною карткою, а Asana також приймає PayPal. У цій статті я припускаю, що ви вирішили виставляти рахунки щорічно, оскільки в іншому випадку ви просто крадете гроші з власної кишені.

Ще одна річ, яку ви побачите, це те, що, як я вже згадував в розділі функцій, спосіб створення планів дуже відрізняється. Це означає, що вам доведеться з'ясувати, що саме вам треба з інструменту управління проектами, перш ніж взяти на себе зобов'язання щодо одного, Отже важливо взяти безкоштовний план для тесту або, ще краще, спробувати 30-ти денну версію. Тим не менш, давайте подивимося на найважливіші моменти і подивимося, чому саме Асана , а не Trello.

3.1.4Зручність використання

Обговоримо зручність для користувачів. Я оголошую цей раунд нічиєю тому що майже всі позитивні моменти: і Asana, і Трелло дуже зручні для користувачів, але давайте подивимося основні моменти.

Trello є одним з найкращих інструментів управління проектами на основі канбану, завдяки простоті використання. Відкриття облікового запису займає всього хвилину або дві - просто заповніть свою адресу електронної пошти, і ви маєте доступ. Є кілька чудових навчальних посібників, які допоможуть вам розпочати роботу, але вони вам ледь потрібні, концепції настільки прості, а додаток Trello настільки інтуїтивно зрозумілий(рис 3.3).

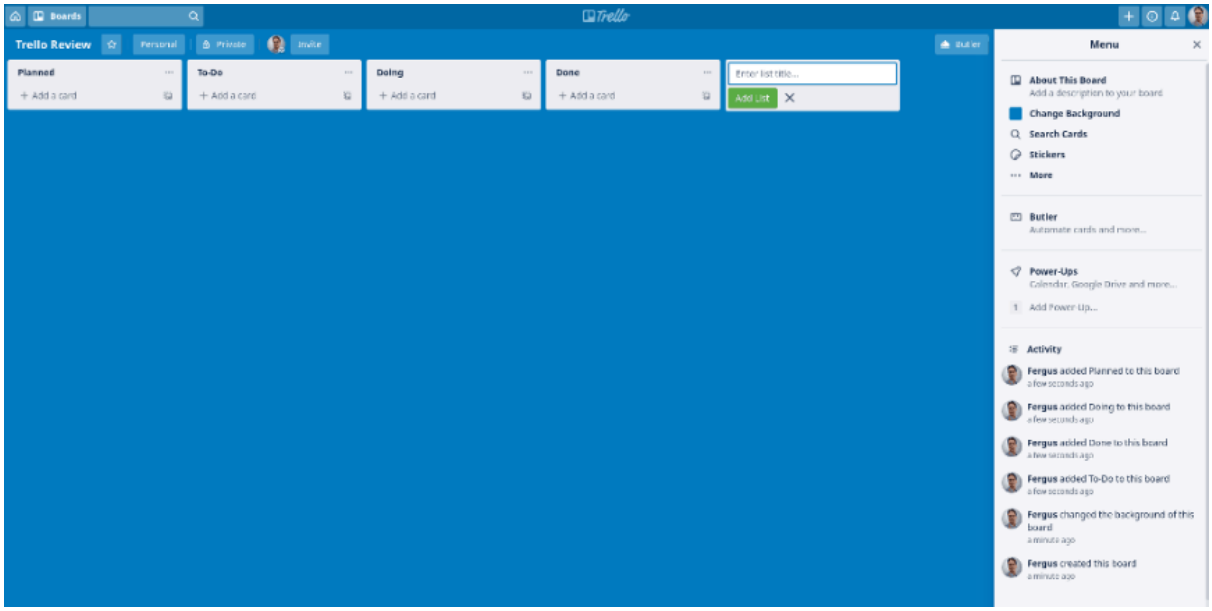


Рисунок 3.3 - інтерфейс Trello

Trello, ймовірно, має найкращі kanban-дошки в бізнесі, і мені подобається огляд, який він пропонує.

Завдання представлені картками , де ви можете додати деталі та етикетки і все це, натиснувши декілька кнопок, які розташовані в колонках. Стовпці Зазвичай представляють завдання, які знаходяться в стадії. Ви перетягуєте картки між

стовпцями, тому перестановка проектів може бути виконана за лічені хвилини, якщо це необхідно.

Trello дуже добре інтегрується з безліччю інших додатків, включаючи популярний додаток для обміну повідомленнями Slack, і є найліпшою і найєлементарнішою системою управління проектами.

3.1.5Висновок:

Наприкінці маємо нічию. Однак, якщо я чесно, функції та ціни, ймовірно, важливіші для середнього менеджера проекту, ніж безпека та підтримка, Отже я впевнений в оголошенні Asana переможцем. Це складне рішення для управління проектами, яке може легко обробляти всі види проектів, великих чи малих.

Тим не менш, є багато причин використовувати Trello, завдяки своїм функціям управління проектами - просто Asana просто робить все це трохи краще. Не має значення, чи використовуєте ви його як програмне забезпечення для управління завданнями або для управління кількома проектами. Єдиний козир, який тримає Трелло, - це його відмінна дошка канбану.

Переможець: Асана

3.2 Trello проти Jira: який інструмент управління проектами кращий?

Trello, швидше за все, є королем канбану, як ви можете прочитати раніше , але не пропонує нічого іншого без використання доповнень. З іншого боку, Jira пропонує kanban і скрам-борд. Крім того, Jira також дає вам деякі інші елементи, а також доповнення; майже такі ж, як і Trello.

3.2.1 Чи Jira тільки для команд розробників програмного забезпечення?

Сьогодні, все ще є аргументи щодо використання команд, які не є програмістами, використовувати Agile управління проектами. Будь-яка група людей, яка працює швидкими темпами або ставить перед собою коротко- та середньострокові цілі, може отримати користь від такого підходу до управління проектами.

Однак, якщо ви розглядаєте Jira через безкоштовні дошки kanban, Trello є набагато кращим варіантом. Давайте почнемо порівнюючи особливості двох наших кандидатів.

3.2.2 Особливості

У першому раунді ми порівняємо функції Jira і Trello. Jira легко виграє його через те, що Trello має дуже мало функцій, однак виграє тут лише через те, що протистоїть такому легкому інструменту управління проектами.

Давайте спочатку подивимося на Trello, який насправді пропонує лише одну функцію у безкоштовному плані, а саме дошку kanban, а також невеликий допоміжний склад для неї. Зрозумійте мене правильно; це одна з найкращих дощок kanban, але вона не робить нічого, крім того, що дозволяє перетягувати списки та картки(рис 3.3).

Допоміжний склад складається з Trello Butler, який допомагає створювати власні робочі процеси, такі як автоматичне призначення термінів при переміщенні картки в певний стовпець тощо, а також можливість додавати мітки та шукати їх та інші характеристики.

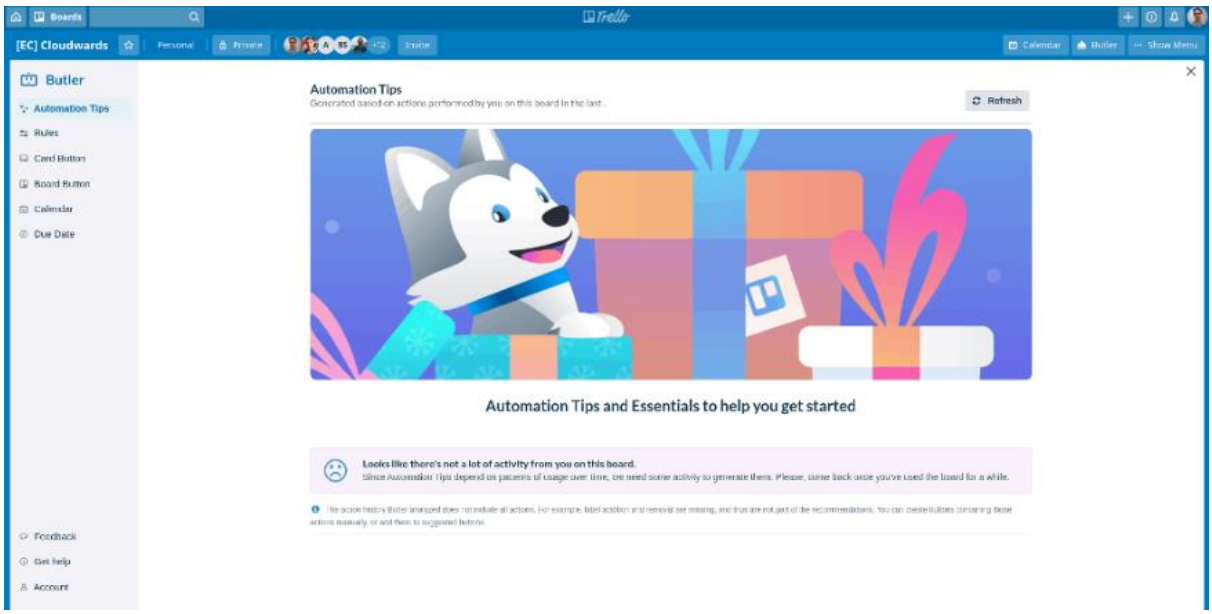


Рисунок 3.4 - Trello Butler

Trello's Butler дозволяє користувачам автоматизувати багато основних завдань, заощаджуючи багато часу.

Крім того, якщо ви вирішите заплатити за Trello, план Premium пропонує інтегрований календар і часову шкалу, а також можливість переглядати завдання на графіку. Я не дуже вражений жодною з цих додаткових функцій, але припускаю, що кілька додаткових функцій керування завданнями ніколи нікому не зашкодять.

Щоб додати щось більше, ніж цю базову функціональність, вам знадобиться використовувати інтеграції або «підвищення», як їх називає Trello, але це заалежить від ваших потреб.

3.2.3Jira: функції для Agile-команд

Jira — або Jira Software, як його чомусь любить називати Atlassian —, був розроблений для програмних студій. Це означає, що він в значній мірі залежить від філософії Agile-розробки, а отже, і від Scrum-борду(рис 3.5), в якій команди працюють Спрінтами з чітко поставленою метою. Коли він зроблений, команда переходить до наступної мети тощо.

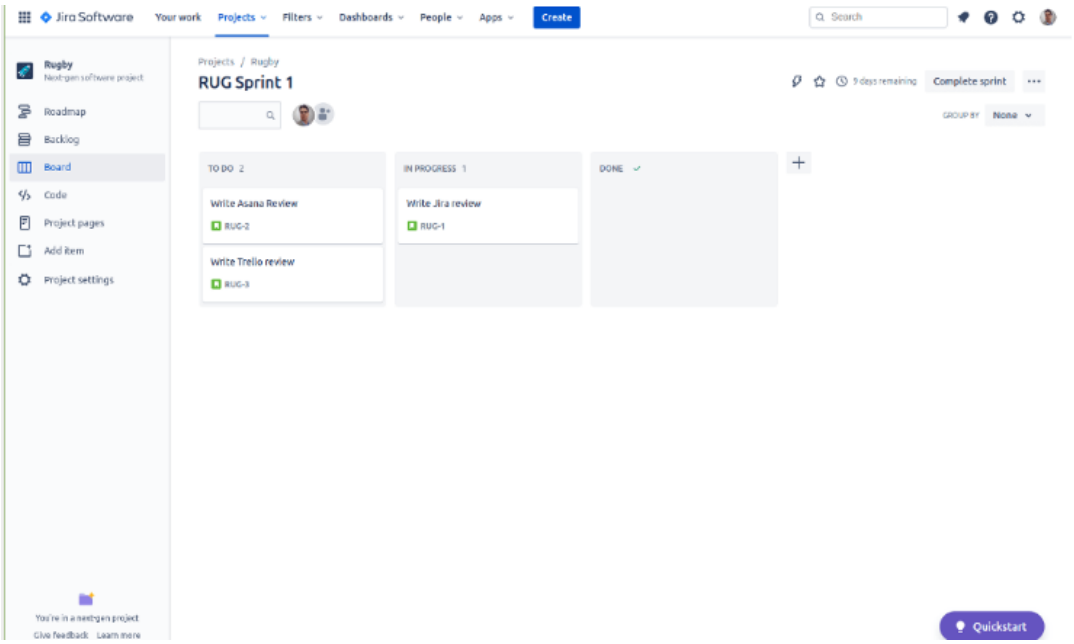


Рисунок 3.5 - Скрам-борд Jira

Скрам-борд є центральним для способу роботи Agile, і у Jira є чудовий.

На щастя для Agile-команд, Jira — одне з найкращих програмних рішень Scrum (Wrike — ще одне). Завдання — або «проблеми», як їх любить називати Jira, — створюються в резерві, а потім без перешкод імпортуються в дошку Scrum. Якщо, крім scrum, ви також хочете використовувати більш лінійний робочий процес, є також дошки kanban(рис 3.6) і дорожня карта(рис. 3.12) для довгострокового планування.

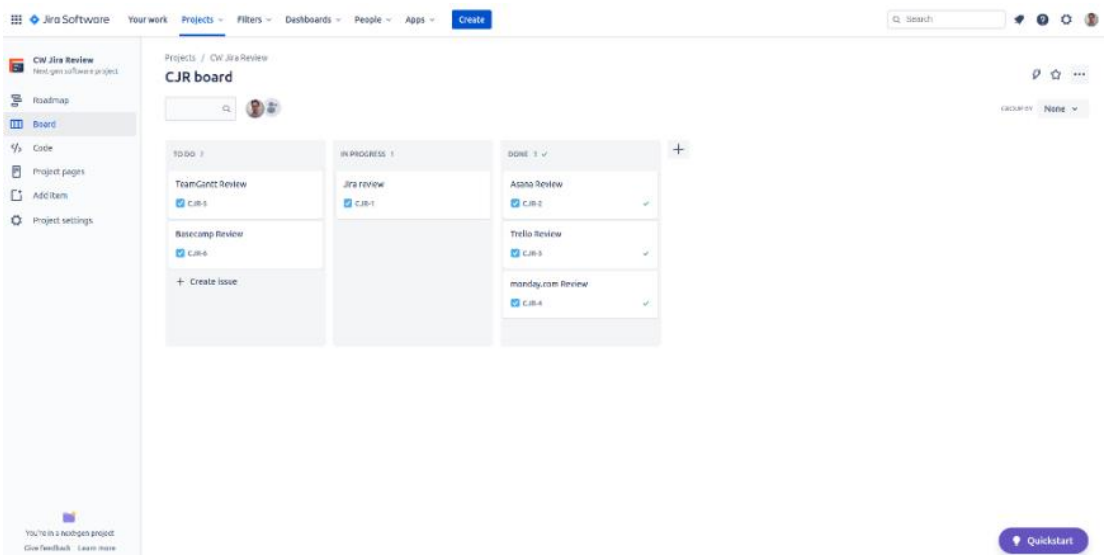


Рисунок 3.6 - Kanban-дошка Jira

Kanban-дошки Jira виконують свою роботу, але це далеко не так корисно, як у Trello.

Як і Trello, Jira має кілька другорядних функцій, які допомагають цим основним. Наприклад, ви можете додавати код та інші ресурси безпосередньо до проекту, що є зручним способом збереження в центральному сховищі під час роботи над проектами.

Ви також можете підключити Jira безпосередньо до Confluence(рис 3.7), який є центром обміну повідомленнями Atlassian і сховищем коду (також є можливість інтеграції Jira-Slack).

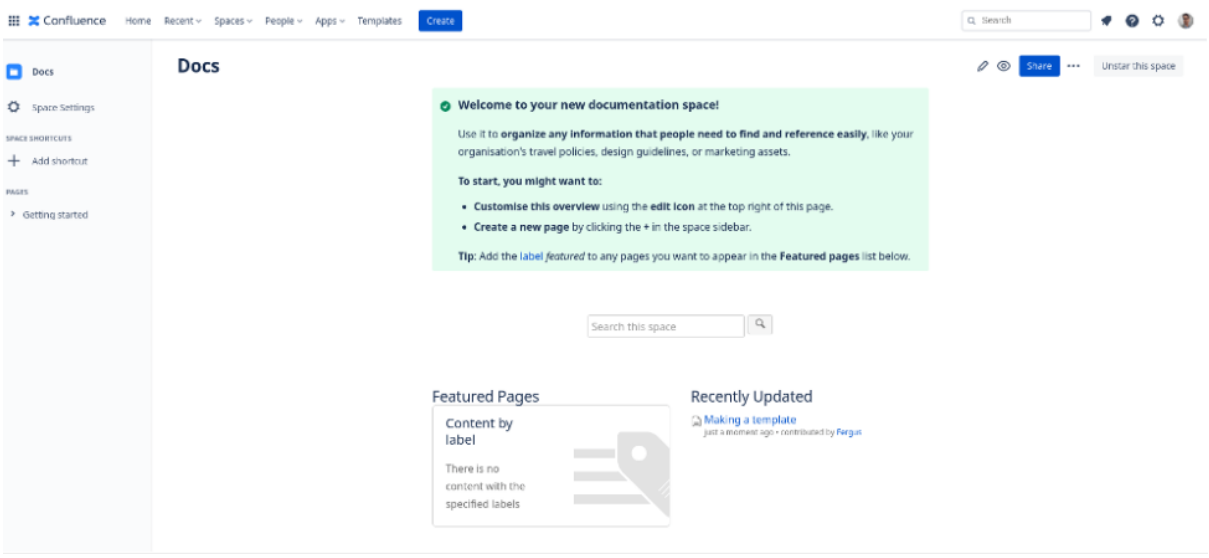


Рисунок 3.7 - Confluence Jira

Confluence, відповідь Jira на Slack, є досить комплексним інструментом, який дозволяє спілкуватися з членами команди, а також планувати завдання та зустрічі.

Це майже все, насправді. Trello має лише kanban-дошки, тоді як у Jira є це, плюс scrum і дорожня карта. Вони обидва мають пристойний допоміжний набір функцій, щоб допомогти цим основним, але насправді, якщо порівнювати з іншими рішеннями для управління проектами, такими як, скажімо, monday.com або Wrike , вони трохи мізерні. Однак тут вступає в дію інтеграція і саме інтеграція іноді маєбільшу вагомість для деяких проектів.

3.2.4Ціноутворення

У першій частині Jira вийшла вперед і ще більше зміцнить свою перевагу в цьому параграфі.

Річ у тім, що обидва, безумовно, є одними з найкращих безкоштовних програм для управління проектами, але платні плани Trello далеко не так конкурентоспроможні, як Jira.

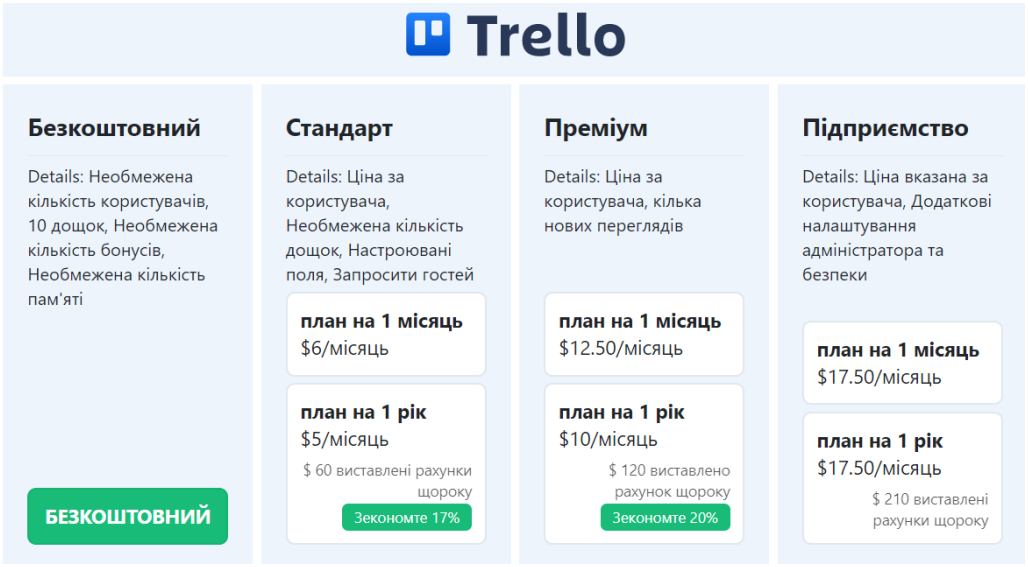


Рисунок 3.1 - Ціни на Trello

Ціноутворення у Trello дуже просте, насправді пропонує лише два плани: безкоштовний і платний . Ми залишаємо план Standard поза увагою, оскільки він дуже обмежений за обсягом, а також Enterprise, оскільки він майже не стосується нашого порівняння в середньом сегменті.

10 доларів на користувача на місяць – це не зовсім дешево, враховуючи те, що ви отримуєте: одну дошка kanban, деякі функції автоматизації та необмежену кількість доповнень, які можуть працювати або не працювати відповідно до вимог. Порівняйте це з Jira, і ви зрозумієте, чому платний план Trello не дуже привабливий.

Рисунок 3.8 - Ціни на Jira

За 7,5 доларів США на користувача на місяць ви отримуєте набагато більше функцій, ніж у Trello. Звичайно, тарифний план Pro коштує 14,5 доларів, тобто дорожче Trello на 40 відсотків, але тоді ви отримаєте функції, про які Trello навіть не мріє.

3.2.5Зручність

Що стосується простоти використання, то тут не так багато чого порівнювати: як Jira, так і Trello дуже прості у використанні. Звичайно, Jira має більше рухомих частин, і його буде важче зрозуміти, ніж інтерфейс перетягування Trello, але Jira компенсує це одними з найкращих навчальних посібників.

Зареєструватися на безкоштовний план будь-якого з них так само просто, як ввести свою адресу електронної пошти, а потім виконати кілька початкових кроків, щоб налаштувати свої дошки.

Як тільки це буде зроблено, ви потрапляєте до вашої дошки, а Jira пропонує вам на вибір і scrum, і kanban (ви не можете мати обидва в одному проекті). Якщо

ви визнали, що ви новачок, Jira запуститься з відкритим меню швидкого запуску(рис. 3.9).

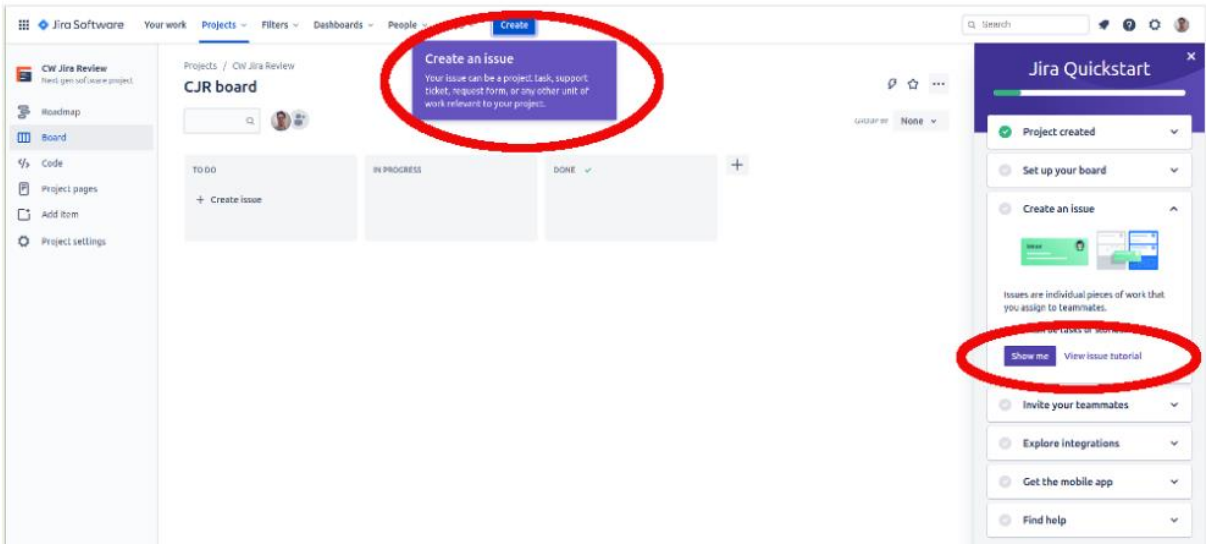


Рисунок 3.9 - Меню швидкого запуску

Jira допомагає новим користувачам використовувати одну з найкращих навчальних систем, які ми бачили.

Мені дуже подобається меню швидкого запуску: це єдина система, яка дозволяє вибирати між переглядом короткого відео на вибрану тему, відвідуванням письмового посібника, отриманням спливаючих вікон на екрані або всіма трьома.

3.2.6Відстеження проблем та інші функції

Приємна особливість kanban і scrum полягає в тому, що вони обидва пропонують широкий огляд, а це означає, що відстежувати завдання та проекти досить легко. Jira була розроблена, щоб зробити це ще простіше, як і Trello, дозволяючи додавати різну інформацію на зворотну сторону карток. Частина цієї інформації підкаже користувачам, що робити із завданням, а частина спростить сортування завдань.

Хоча спочатку додавання цих деталей до картки може здатися обтяжливим, миттєво ви звикнете до цього. Ще одна перевага полягає в тому, що всю цю інформацію можна отримати як дані — звичайно, використовуючи правильні інтеграції — і потім використовувати для подальшого огляду за допомогою діаграм тощо.

Загалом, скаржитися тут дуже мало на що. І до Jira, і до Trello до смішного легко звикнути та використовувати.

3.2.7Висновок:

Ось так, впевнена перемога Jira . Насправді це не є великим сюрпризом: як я казав у вступі, Jira просто має перевагу, коли справа доходить до функцій і ціни.

Тим не менш, у будь-якій системі, яка використовує дошку kanban, є місце для Trello, і ніщо не рятує вас від інтеграції цих двох.

Переможець: Джира

3.3 Асана проти Джира: розбір інструментів управління проектами

3.3.1Особливості

Наш перший раунд – легка перемога для Асани. Це один з найбільш багатофункціональних інструментів управління проектами, який навіть перевершує monday.com у кількох ключових областях, отже це найкраща альтернатива для шанувальників Asana.

Це не означає, що Jira не годиться; просто її материнська компанія, Atlassian, вирішила застосувати набагато більш спрощений підхід. Для цього є дві причини: перша полягає в тому, що філософія Agile, на якій базується Jira, є дуже спрощеним методом, який не вимагає зайвих дій для виконання роботи.

По-друге, Atlassian любить пропонувати просте програмне забезпечення, яке користувачі потім можуть налаштувати відповідно до власних потреб. Приблизно те саме відбувається з Trello, братом Jira, де за певних обставин сторонні інтеграції можуть в кінцевому підсумку становити більшу частину програми, ніж її ядро.

3.3.2Особливості Jira

Давайте подивимося на основні функції Jira. Зрозуміло, це не більше ніж дошка scrum(рис 3.11) або kanban(рис 3.10) (будь-який проект може мати лише один з них), а також кілька допоміжних акторів, як-от сховище коду.

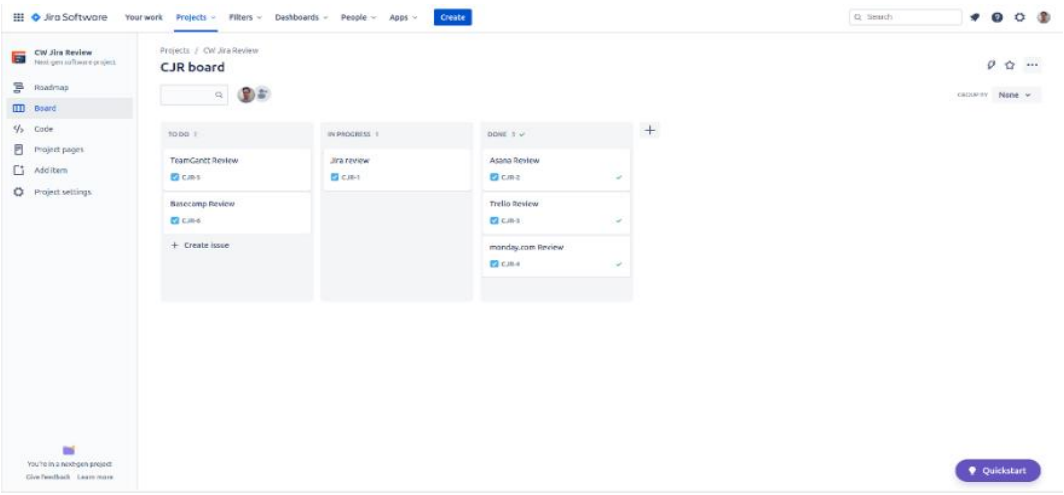


Рисунок 3.10 - Канбан-дошка Jira

Канбан-дошка Jira виконує свою роботу, хоча вона не настільки повнофункціональна, як у Asana або навіть Trello.

Знову ж таки, це не оціночне судження: якщо це працює, то працює. Користувачі створюють картки, які представляють завданням (або «проблемою», кажучи мовою Jira), потім кладуть їх на дошку канбан, готові до перетягування. Це хороший спосіб підтримувати регулярний робочий процес.

Команда розробників програмного забезпечення, швидше за все, витрачатиме свій час на дошку Scrum, де замість лінійного набору завдань ви встановлюєте «спринти» — завдання, які треба виконати за короткий проміжок часу. Ви налаштовуєте спринт, спочатку створюючи проблеми в резерві, а потім імпортуючи їх на дошку Scrum.

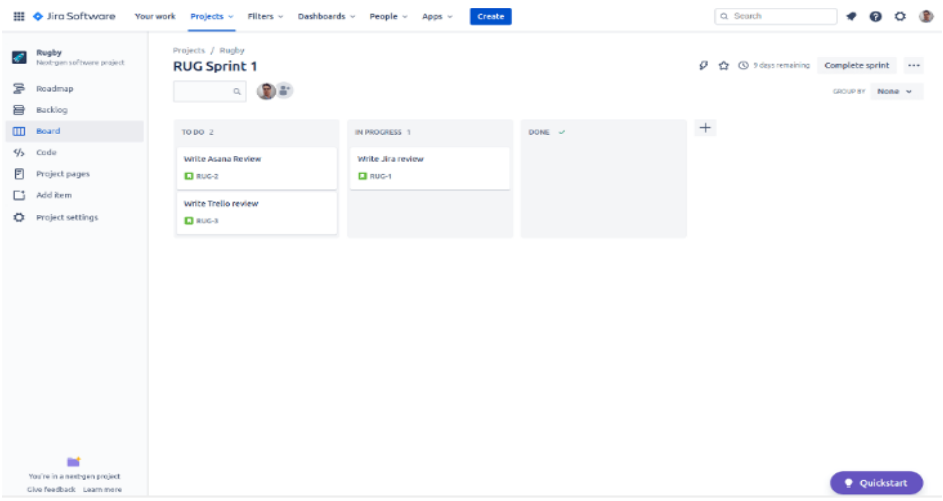


Рисунок 3.11 - Scrum board Jira

Scrum board Jira — це простий, але ефективний інструмент, особливо якщо він використовується разом із допоміжними функціями.

Agile-команди зазвичай працюють від спринту до спринту, поки проект не буде завершено. Сам scrum не дасть особливого огляду, але Jira додала так звану дорожню карту, яка дозволить вам відстежувати загальний прогрес проекту. Мені подобається ця функція, але я б хотів, щоб було більше способів контролювати використання Jira; як наприклад зробили Wrike.

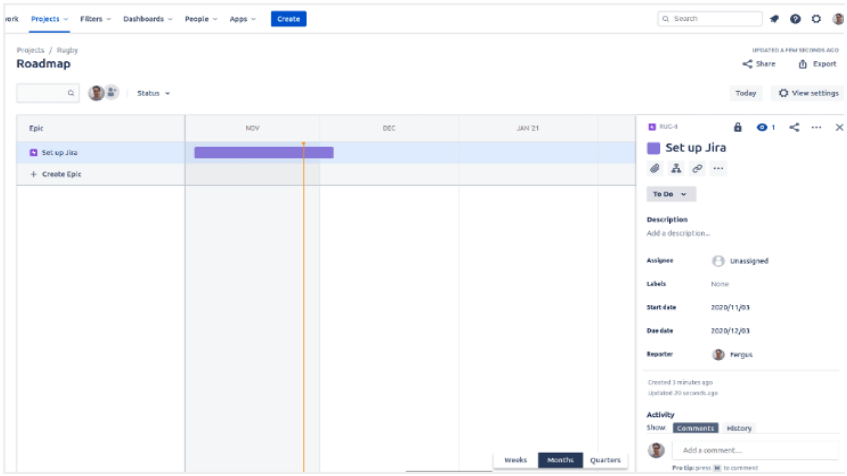


Рисунок 3.12 - Дорожня карта Jira

Дорожня карта Jira — це єдиний спосіб отримати загальне уявлення про ситуацію

3.3.3Можливості Asana

Якщо Jira орієнтована на розробку програмного забезпечення, Asana є скоріше загальним інструментом керування завданнями, який можна використовувати для будь-якого проекту. Він пропонує багато стандартних функцій, які ви знайдете, скажімо, на monday.com, але ви також можете створювати власні функції — у розумних межах — щоб задовольнити будь-які конкретні потреби.

Але спочатку давайте трохи поговоримо про його повсякденне використання. Користувач може працювати в кількох середовищах, які називаються «поданнями», включаючи список, канбан, календар і (якщо ви оновлюєте) часову шкалу, а також кілька інших. Це може здатися складним, але насправді це дуже просто: ви створюєте завдання(рис 3.13), скажімо, у вигляді списку (що я рекомендую), а потім представлення є лише способами їх організації.

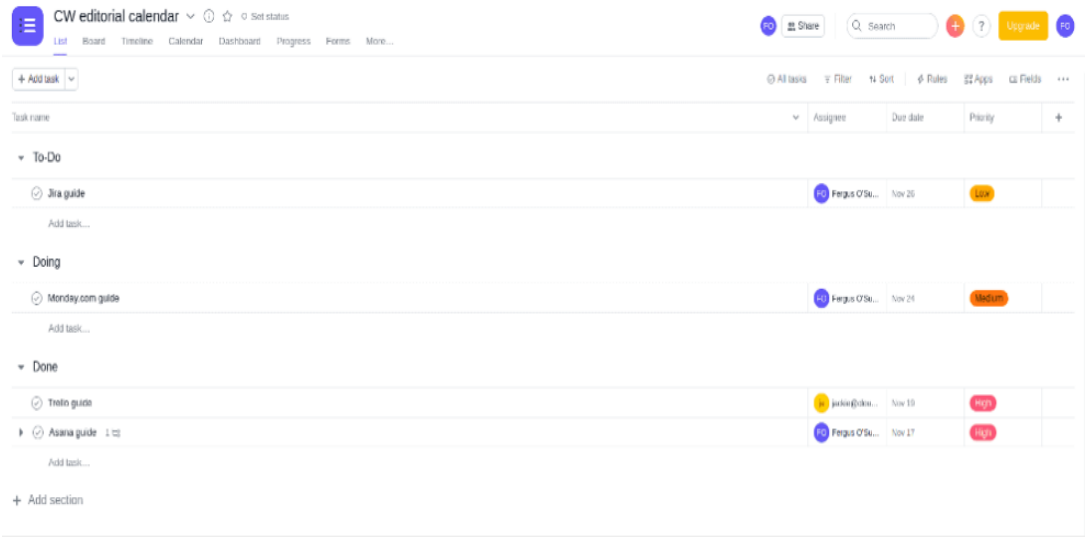


Рисунок 3.13 - Створення завдання Asana

Перегляд списку, ймовірно, є тим, який використовуватиметься за замовчуванням в Asana, оскільки він пропонує найбільше опцій.

Наприклад, ви вводите набір завдань у свій список, вказуєте їм терміни виконання, пріоритетні наклейки та статус. Після цього ви можете перейти до перегляду kanban,(рис 3.14) щоб побачити статус усіх ваших карток, діаграми,

щоб визначити, скільки завдань і який пріоритет, а потім перегляду календаря, щоб побачити, коли все закінчиться.

Asana надає велику силу та контроль при правильному використанні, і в цьому відношенні вона «вибиває Jira з води».

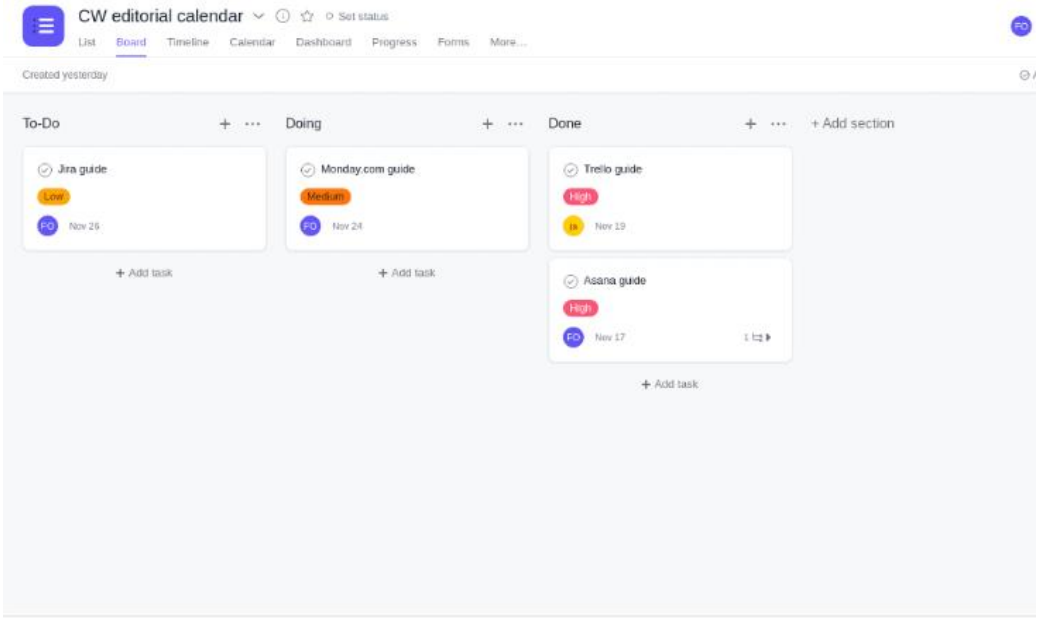


Рисунок 3.14 - Канбан-дошка Asana

Канбан-дошка Asana чудово підходить, якщо ви хочете планувати й відстежувати завдання, і вона приємніша, ніж у Jira.

Це також найбільша різниця між Jira та Asana: Jira насправді має лише два перегляди на проект. Під час створення великого проекту єдиний хороший спосіб отримати будь-який нагляд — це дорожня карта, яка є хорошою, але не фантастичною. У Asana є кілька способів побачити не тільки те, що відбувається зараз, але й будь-який розумний час у майбутньому(рис 3.15).

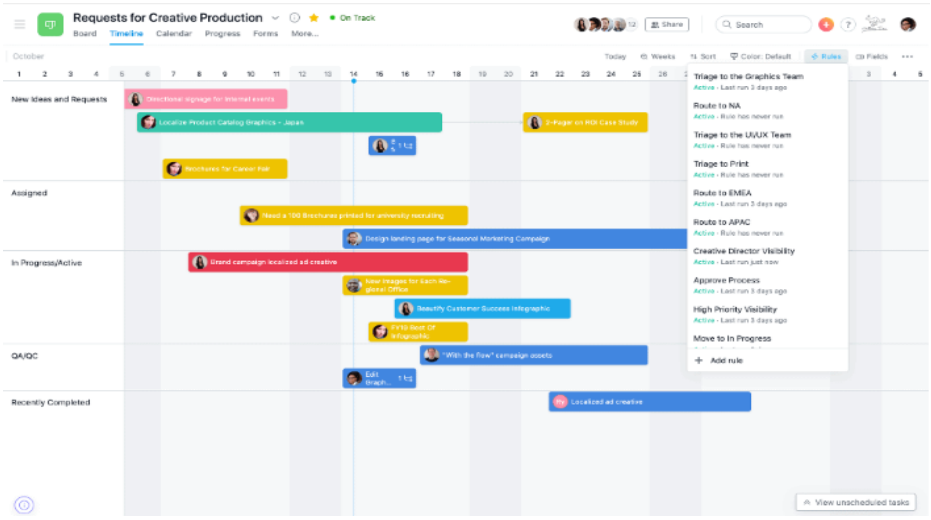


Рисунок 3.15 - Часової шкали Asana

Перегляд часової шкали Asana допомагає вам визначити тривалість завдань, що допомагає ефективніше розподіляти ресурси.

3.3.4Управління членами команди

У Асані також набагато краще управління членами команди. Jira, звичайно, дозволяє призначити кого завгодно до вашої дошки (просто введіть їх ім’я та електронну адресу — ви також можете додати зображення), але фільтрація за членами команди дає вам лише завдання, призначені їм.

Для порівняння, Asana може показати вам, що вони роблять, коли, а також порівняти робоче навантаження окремих членів команди(рис 3.16) один з одним за допомогою однойменного перегляду.

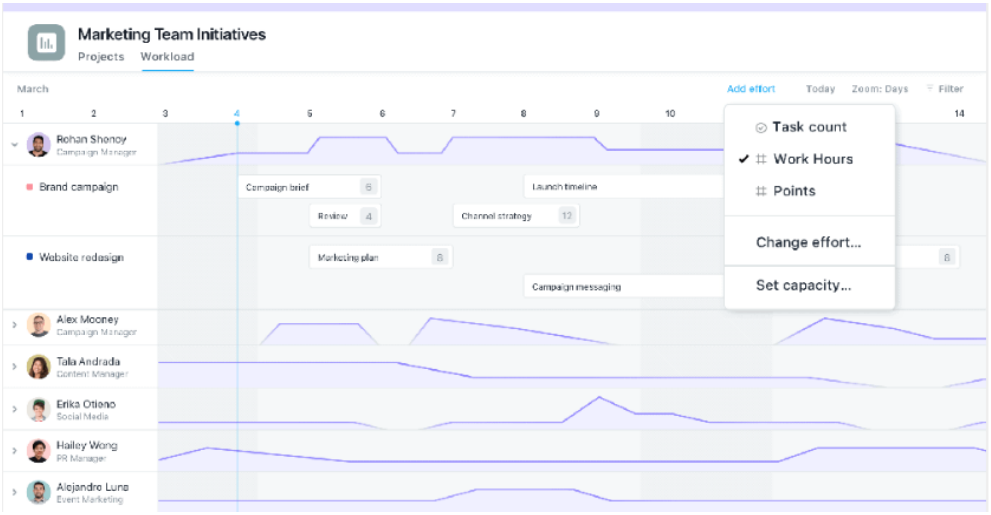


Рисунок 3.16 - Робоче навантаження Asana

Перегляд робочого навантаження Asana дає змогу побачити, чи хтось працює занадто важко або, навпаки, не тягне свою вагу.

Звичайно, це лише найважливіші характеристики — їх ще багато. Наприклад, є вбудований додаток для відстеження часу під назвою Harvest, а також перегляд портфоліо, що дозволяє керувати «з висоти пташиного польоту» кількома проектами одночасно. Цей останній ідеально підходить для менеджерів проектів, які самі керують менеджерами.

Однак є дві речі, які слід пам’ятати, порівнюючи особливості Asana та Jira. По-перше, Jira відкриває більшість здібностей у безкоштовному плані, тоді як Asana ревно охороняє свої функції за кількома рівнями оплати. По-друге, Jira може багато чого зробити за допомогою інтеграцій.

3.3.5Ціноутворення

Оскільки Асана виходить ізтвердою перемогою, я збираюся перейти до менш результативного раунду. Справа в тому, що Jira пропонує набагато кращі ціни в доларах і центах, але ми вважаємо, що вартість Asana набагато краща(рис 3.18),

якщо ви подивитеся, скільки ви отримуєте за свої гроші. Щоб пояснити, нам треба трохи поринути в суть. Почнемо з цін Jira(рис 3.17).

Рисунок 3.17 - Ціни Jira

Як бачите, Jira просто дешева. Якщо ви навіть вирішите заплатити за Jira: невеликі команди розробників з менш ніж 10 користувачами могли б використовувати Jira безкоштовно без обмежень, оскільки всі дійсно життєво важливі функції включені в безкоштовний план. Асана могла б навчитися цього, але про це пізніше. Поки обговоримо лише платні плани.

Якщо ви перейдете до стандартного плану, Jira коштує лише 7,5 доларів на користувача на місяць, а преміум-план — ще 7,5 доларів США, що становить 14,5 доларів (за умови, що ви платите на рік, що вам треба в більшості випадків).

Це дешево за будь-яким показником, але порівняйте це з іншими, monday.com проти Jira , і результат може бути шокуючим. Для порівняння давайте подивимося, як виглядають ціни на Asana.

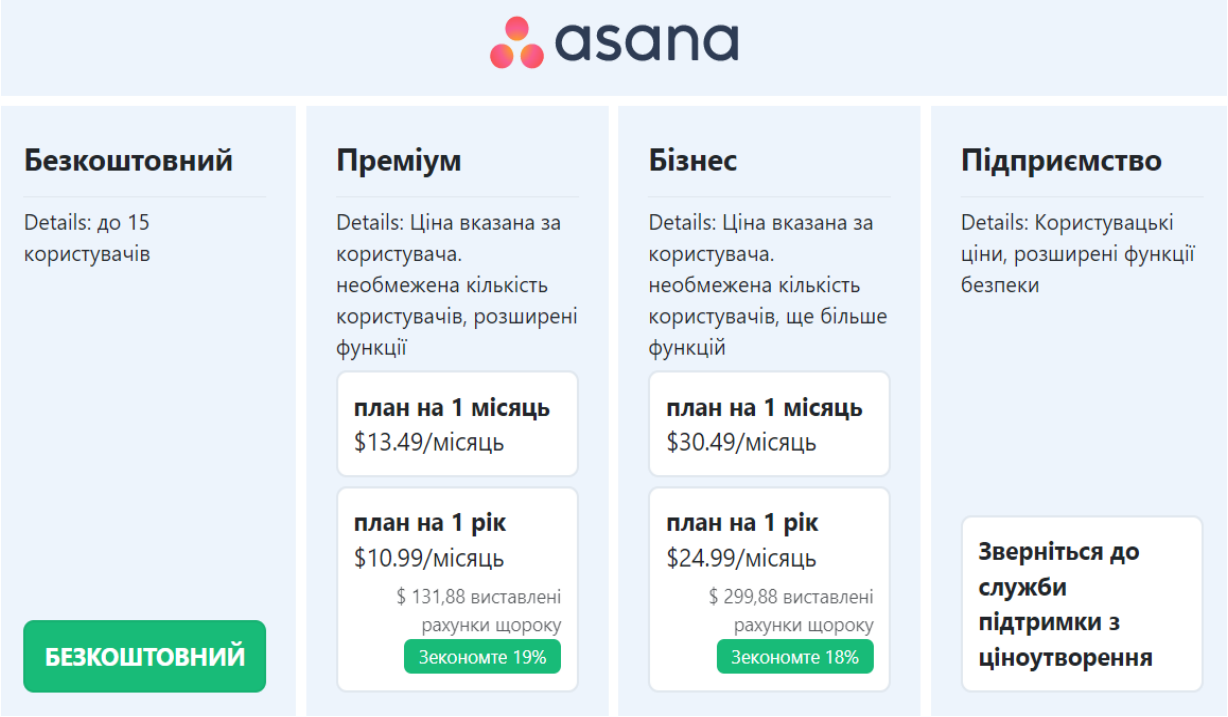


Рисунок 3.18 - Ціни Asana

Це досить різкий контраст, і це навіть не враховуючи безкоштовний план. Преміальний план Asana становить 11 доларів США на користувача на місяць (за умови річної підписки), що на 4,5 бакса більше, ніж стандартний план Jira. Крім того, бізнес-план Asana коштує колосальних 25 доларів на користувача на місяць, що на 11 доларів більше, ніж преміальний план Jira.

Я залишаю корпоративні плани поза формулою, оскільки в обох випадках вони пропонують дуже специфічні та спеціалізовані функції, здебільшого пов’язані з безпекою та дозволами користувача.

3.3.6Зручність Використання

Поки що Асана все ще попереду, але це змінюється в цьому раунді, оскільки Джира отримує зрівняльний бал. Хоча Asana проста у використанні, Jira просто легше. Багато в чому це завдяки панелі швидкого запуску, яка є однією з найкращих функцій Jira. Насправді, хотілося б, щоб більше інструментів управління проектами було.

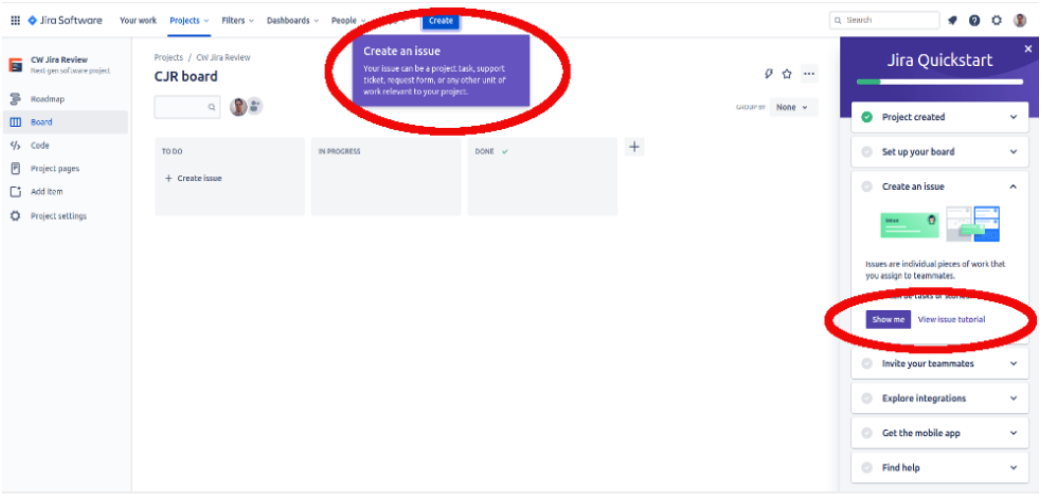


Рисунок 3.19 - Меню швидкого запуску Jira

Панель швидкого запуску Jira надає вам кілька варіантів, щоб краще ознайомитися з програмним забезпеченням Jira.

Коротше кажучи, швидкий Початок(рис 3.19) – це панель у правій частині екрана, яка направляє вас через найпростіші завдання, дозволяючи вибирати між отриманням спливаючих вікон на екрані, переглядом відео, читанням документації або всіма трьома.

На відміну від цього, Asana використовує лише спливаючі вікна, хоча ви можете вивчати свій шлях через величезну базу знань Asana. Взагалі кажучи, спливаючі вікна виконують свою роботу, хоча іноді поради здаються дещо недоречними.

3.3.7Висновок:

Ось і все, перемога Асана. Врешті-решт саме відсутність функцій довела до краху Jira. Тим не менш, Jira виступила проти одного з найкращих інструментів управління проектами, що більше, ніж можуть зробити багато інших продуктів на ринку.

Переможець: Асана

3.4 Систематизування аналізу

Для того, щоб програмне забезпечення допомагало працювати над проектом, а не ускладнювало процес необхідно, щоб воно було зручне у використанні як за інтуїтивністю інтерфейсу, так і за багатьма іншими функціями. Існують такі програми як Trello [1], Asana [2], Jira [3], Oracle Primavera [4] та інші. Всі ці програми постійно вдосконалюються у тому, щоб споживачеві було зручніше у яких працювати [1-5].

На ринку програмного забезпечення величезний вибір програм для управління проектами. Можна знайти як безкоштовні версії, так і платні. Основна їх відмінність у великій кількості інструментів для роботи. Були обрані найбільш популярні програми з усієї множини. Проаналізувавши більше 30 програмних забезпечень, було складено список, що налічує 12 платформ(детальне порівняння одне з одним у минулому розділі), найбільш функціональних з них.

Були розглянуті такі програми:

- Jira ;
- GanttPro ;
- BaseCamp ;
- ActiveCollab ;

• Wrike ;

• Smartsheet ;

• MS Project ;

• Redbooth ;

• Trello ;

• Assana;

• Odoo ;

• Oracle Primavera .

Аналіз та порівняння програм проводилися за такими параметрами:

• Можливість роботи онлайн та наявність хмарного зберігання даних;

• Спільний доступ;

• Наявність діаграми Гантта ;

• Відстеження розвитку проектів, у тому числі у відсотках;

• Налаштування фільтрів;

• Синхронізація з іншими програмами;

• Можливість чату та обміну файлами всередині програми;

• ОС, які підтримує програма;

• Подання інформації у вигляді діаграм;

• Створення звітів за проміжок часу від початку проекту;

• Онлайн повідомлення;

• Простота інтерфейсу;

• Облік витрат на певний час;

• Можливість автоматично отримувати прогнози розвитку проекту;

• Наявність пробного періоду, безкоштовний тариф;

• Вартість ліцензійної версії програми.

У сучасному суспільстві величезний потік інформації йде через інтернет, тому можливість роботи онлайн та зберігати дані у хмарі дуже цінується у багатьох програмах. Ця функція дозволяє з будь-якої точки

світла, де є підключення до мережі, вносити якісь корективи в проект, контролювати його виконання, поки перебуваєш у відрядженні або відпустці.

Спільний доступ до програми дозволяє спростити командну роботу, більше не потрібно переміщати всі дані з одного комп'ютера на інший та поєднувати їх.

Діаграма Ганта є невід'ємною частиною програм для управління проектами. Вона являє собою стовпчасту гістограму, яка ілюструє графік роботи з якогось проекту.

Відстеження розвитку проекту, як у частках, і у відсотках, дозволяє аналізувати з випередженням чи відставанням за графіком розвивається проект. Дуже зручна функція. Особливо для масштабних проектів, коли на коні є великі фінансові втрати.

Для того, щоб знайти якусь інформацію швидко, не переглядати її по всьому проекту в пошуках необхідної, приходить на допомогу така функція, як налаштування фільтрів відображення.

Синхронізація з іншими ПЗ дозволяє безперешкодно переносити інформацію з однієї машини на іншу, що так само прискорює роботу над проектом, дозволяє учасникам проекту отримувати будь-які повідомлення у своїх гаджетах, щоб не пропустити важливий захід, наприклад, дедлайн.

Можливість чату та обміну файлами всередині програми зручна тим, що не потрібно встановлювати додаткові програми.

Ще одним важливим пунктом огляду програм для УП є те, в яких ОС вона може працювати. Що більше ОС підтримується, то більше людей її використовуватиме.

Подання інформації як діаграм дозволяє спростити її сприйняття.

Оскільки були проаналізовані програми з ухилом на російського споживача, наявність російськомовної версії є перевагою.

Створення звітів за будь-який проміжок від початку проекту дозволяє вести звітну документацію та стежити за розвитком проекту.

Простота або інтуїтивність інтерфейсу хороша тим, що дозволяє непросунутому користувачеві досить швидко освоїтися в програмі і почати відразу в ній працювати, а отже, не потрібно відправляти працівника на тривале навчання.

Облік витрат за певний час показує скільки коштів було вкладено у проект, скільки залишилося вкласти, чи працює підприємство на збиток чи навпаки економніше. Звідси випливає можливість автоматичного прогнозування, що також дуже зручно.

Наявність пробного періоду або безкоштовної версії дозволяє користуватися в навчальних цілях програмою або підібрати найбільш підходящу, якщо з незручно працювати. Наявність безкоштовної версії є величезним плюсом для малого бізнесу.

Після закінчення пробного періоду постає питання покупки ліцензійної версії програми. Чим більша компанія, тим дорожчу програму вона може собі дозволити.

Результати аналізу ПЗ представлені у таблиці 1.

Для оцінки програм була розроблена шкала, в якій кожному параметру зіставили кілька балів, а підсумковим балом програми стає сума балів, отриманих програмою за кожен параметр.

Бали були розподілені так:

• "Онлайн": мінімальний бал - 0, максимальний бал - 1;

• Спільний доступ: 0-2;

• Наявність діаграми Ганта : 0-1;

• Відстеження розвитку проектів: 0-1;

• Налаштування фільтрації: 0-2;

• Синхронізація з іншими програмами: 0-2;

• Можливість чату: 0-1;

• Підтримка ОС: 0-6;

• Обмін файлами: 0-2;

• Візуалізація даних: 0-1;

• Створення звітів: 0-2;

• Сповіщення: 0-1;

• Інтуїтивність інтерфейсу: 0-1;

• Відсоток виконання завдань: 0-1;

• Облік витрат: 0-2;

• Прогнози: 0-2;

• Хмарне сховище: 0-2;

• Наявність пробного періоду: 0-1;

• Безкоштовний тариф: 0-1;

• Вартість: 1-5.

У більшості випадків програма отримує найвищий бал за параметр, якщо він є, а мінімальний бал – за відсутність. Винятки становлять «Вартість» та «Підтримка ОС».

У параметрі «Вартість» привласнення балів велося так: всі ціни були приведені доларовій. Далі всі ціни були розділені на кілька категорій, кожній категорії ставиться у відповідність бал: найдешевшої категорії – максимальний бал, а найдорожчий – мінімальний. Бали розподілені так:

• До 800 грн/місяць – 5 балів;

• Від 801/місяць до 1300/місяць – 4 бали;

• Від 1301/місяць до 1900/місяць – 3 бали;

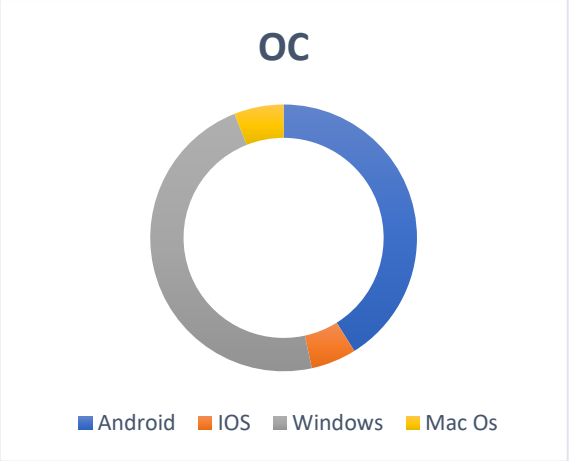
• Від 1901/місяць до 2500/місяць – 2 бали;

• Від 2501/місяць – 1 бал.

Валюта за наближеними курсами: 31 гривня за 1 долар США, 34 гривнії за 1 євро

Для виставлення балів за параметром "Підтримка ОС" було проаналізовано кількість користувачів кожної операційної системи, знайдено відсоток користувачів по кожній системі від загальної кількості. За кожні 20% користувачів надається 1 бал. Максимально можна здобути 5 балів. Також 1 бал додався за можливість працювати в інтернеті (WEB).

Разом за цей параметр можна отримати 6 балів. Відсоткове співвідношення користувачів різних ОС представлено кругової діаграмі 3.



Діаграма 3 - Співвідношення користувачів різних ОС

Результати розбалування та рейтинг програм за загальною кількістю балів представлені у таблицях нижче(рис 3.20, 3.21).

Одним із головних параметрів є вартість програми. Найбільш низькобюджетними є RedBooth , Trello та MS Project, Asana. Але для виконання невеликих проектів можна використовувати програми з безкоштовним тарифом, яким володіють: GanttPro , BaseCamp , Wrike , Trello, Asana , Odoo , Oracle Primavera . Так само кожна програма має пробний період користування, що дозволяє визначитися з підходящою програмою для тих чи інших завдань.

До трійки лідерів увійшли:

• Asana – 34,9 балів;

• Oracle – 31,6 балів;

• MS Project – 32,4 бали.

	Jira	Gantt Pro	BaseCamp	ActiveCollab	Wrike	Smartsheet	MS Project	Redbooth	Trello	Asana	Odoo	Oracle Primavera
Online	+	+	-	+	+	+	+	+	+	+	+	+
Суспільний доступ	+	+	+	+	+	+	+	+	+	+	+	+
Діаграма Ганта	+	+		+	+	+	+	+	+	+	+	+
Відстежування розвитку проекту	+	+	+	-	-	+	+	-	+	+	-	+
Налаштування фільтрації	+	-	-	+	-	+	+	-	+	+	-	+
Синхронізація зі стороннім ПО	+	+	+	+	+	+	+	+	+	+	+	+
Можливість чату	+	-	+	-	+	-	-	+	+	+	+	-
Підтримка ОС	Android Windows Mac Web	Windows Mac Web	Android Windows Mac Web	Android Web	Android Web	Android Web	Windows Web	Android Web	Android Windows Mac Web	Android Web	Android Windows Mac Web	Android Windows Mac Web
Обмін файлами	+	+	+	+	+	+	+	+	+	+	+	+
Візуалізація даних	+	+	-	-	-	+	+	+	+	+	+	+
Створення звітів	+	-	+	+	+	+	+	+	-	+	+	+
Система сповіщення	+	+	+	-	+	+	-	+	+	+	-	-
Інтуїтивність інтерфейсу	+	+	+	-	+	-	+	-	+	+	+	+
Процент виконання задач	+	-	-	-	+	-	+	-	+	+	-	+
Ведення витрат	-	-	-	+	+	+	+	-	-	+	+	+
Прогнози	-	+	-	-	-	-	+	-	-	+	-	+
Хмарне сховище	+	+	+	+	+	+	+	+	+	+	+	+
Пробний період	+	+	+	+	+	+	+	+	+	+	+	+
Безкоштовний тариф	-	+	+	-	+	-	-	-	+	+	+	+
Вартість	10$	20$	100$	25$	10$	10$	9$	5$	10$	25$	15$	52,99$

Рисунок 3.20 - Таблиця наявності функцій

	Jira	Gantt Pro	BaseCamp	ActiveCollab	Wrike	Smartsheet	MS Project	Redbooth	Trello	Asana	Odoo	Oracle Primavera
Online	1	1	0	1	1	1	1	1	1	1	1	1
Суспільний доступ	2	2	2	2	2	2	2	2	2	2	2	2
Діаграма Ганта	1	1	0	0	1	1	1	1	1	1	1	1
Відстежування розвитку проекту	1	1	0	0	1	1	1	1	1	1	0	1
Налаштування фільтрації	2	0	0	0	0	2	2	0	1	2	0	1
Синхронізація зі стороннім ПО	2	2	2	2	2	2	2	2	2	2	2	2
Можливість чату	1	0	1	0	1	0	0	1	1	1	1	0
Підтримка ОС	5,6	1	5	3,4	3,4	3,2	3,4	5,5	5	5,9	3,4	5,6
Обмін файлами	2	2	2	2	2	2	2	2	2	2	2	2
Візуалізація даних	1	1	0	0	1	1	1	1	1	1	1	1
Створення звітів	2	0	2	2	2	2	2	2	0	2	2	2
Система сповіщення	1	1	1	0	1	1	0	1	1	1	0	0
Інтуїтивність інтерфейсу	0	1	1	0	1	0	1	0	1	1	1	1
Процент виконання задач	1	0	0	0	1	0	1	0	1	1	0	1
Ведення витрат	0	0	0	2	2	2	2	0	0	2	2	2
Прогнози	0	2	0	0	0	0	2	0	0	2	0	2
Хмарне сховище	2	2	2	2	2	2	2	2	2	2	2	2
Пробний період	1	1	1	1	1	1	1	1	1	1	1	1
Безкоштовний тариф	0	1	1	0	1	0	0	0	1	0	1	1
Вартість	5	5	1	4	5	5	5	5	5	4	5	3
Бали	30,6	24	21	21,4	30,4	28,2	31,4	27,5	29	34,9	27,4	31,6
Місце в рейтингу	5	10	12	11	3	7	4	8	6	1	9	2

Рисунок 3.21 - Таблиця оцінки ефективності ф-цій

3.5 Висновоки до розділу:

В результаті проведенної роботи було узагальнено критерії оцінки першості існуючих рішень за кількість та ступенем реалізації функціоналу, а також винесено вирок доцільності створення власного продукту, та рекомендований алгоритми створення підсистеми, що могла б покращити існуючі рінення, або зробити власний проект управління версіями конкурентно спроможним.

4 РЕКОМЕНДАЦІЇ ЩО ДО МОДИФІКАЦІЇ ВЕБ-ОРІЄНТОВАНОГО СЕРЕДОВИЩА «УПРАВЛІННЯ ІТ ПРОЕКТАМИ»

Аналітична підсистема веб-орієнтованого повинна бути доступною, масштабованою та зручною як в користувацькіій частині так і в плані загальної розробки. В аналітичній підсистемі повинна бути гнучка архітектура, яка дозволила б масштабувати операції для запису та зчитування. Інтерфейс аналітичної підсистеми повинен мати елементи управління, що дозволять фільтрувати статистику користувачів за різними параметрами.

Система повинна відповідати цим вимогам:

- Велика швидкодія;
- Можливість перегляду результатів тестів;
- Розподіленняя статистики по різними критеріями;
- Видалення статистики;
- Ф-ція перегляду кількості вірних відповідей діаграми;
- Мати захищене з’єднання;
- Пошук користувачів за логіном;
- Перегляд результату діаграми;
- Ф-ція ранжування прав;

4.1 Структура інформаційної системи

Змоделюймо її архітектуру системи, щоб зрозуміти як вона працює. Частину фронтенд буде спроектовано шляхом зменшення внесення змін в панель користувача та адміністратору. Загальна бібліотека буде основною службою для постачання основного функціоналу і використання модулю статистики.

За допомогою версій контролю коду Github на локальному диску зберігаються репозиторії для панелєй користувача, адміністратору, бекенду та загальна бібліотека(рис 4.2) для фронтенду(рис 4.1). Спочатку створиться сторінка для статистики користувача, а потім для адміністратора. Компіляція модуля статистики залежить від загальної бібліотеки, без неї робота неможлива.

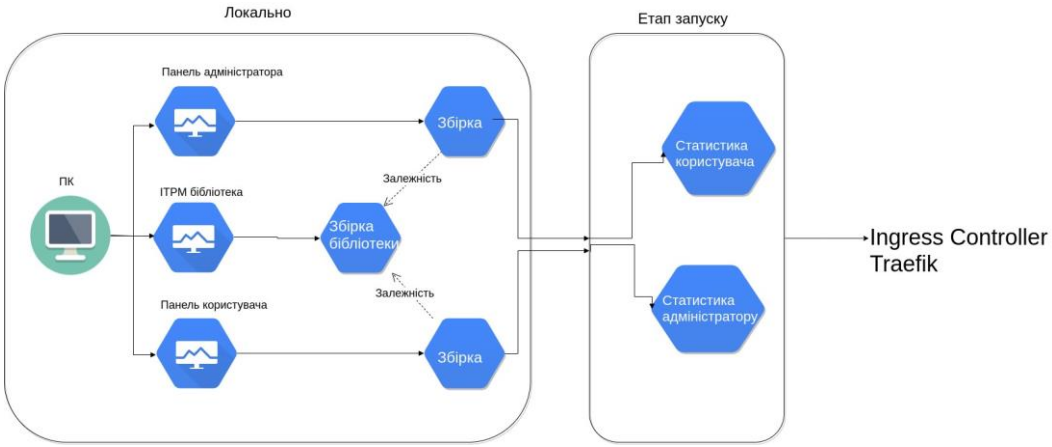


Рисунок 4.1 - Фронтенд архітектура системи

Бекенду архітектура спроектована на великі навантаження з боку зчитувальних операцій та операцій запису статистики в базу даних. При великому збільшенні кіль-ті підключень до бекенду, кількість його «інстансів» буде збільшено при досяганні ліміту використання пам’яті на 80% ,а утилізації процесору відбудеться на 50% відповідно.

Масштабування внутрішньої частини проекту, а саме баз даних для статистичних даних буде відбуватися за допомогою Docker.

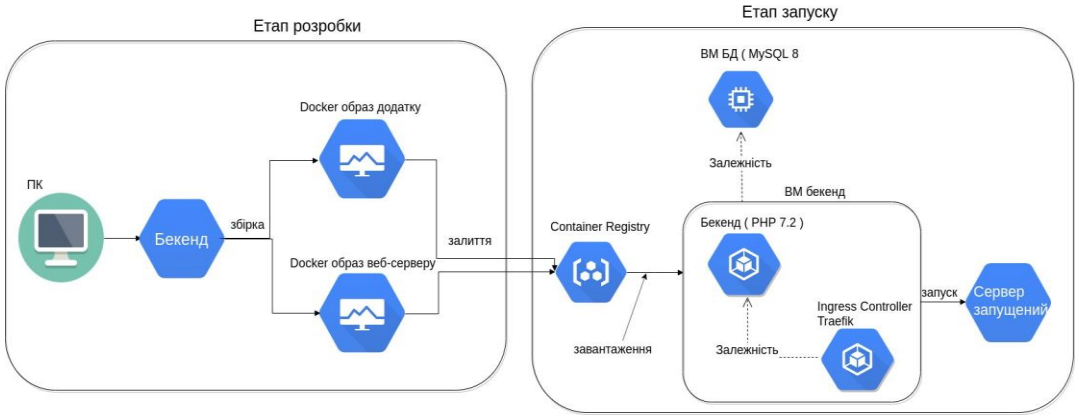


Рисунок 4.2 - Бекенду архітектура

Docker - це проект з відкритим кодом для автоматизації розгортання додатків. Виглядає як набір автономних контейнерів(рис 4.3), які виконуються в хмарі або локальному середовищі. Також, Docker - це компанія, яка розробляє і просуває цю технологію у співпраці з постачальниками «хмарних» послуг, а також рішень Linux і Windows, включаючи корпорацію Майкрософт.

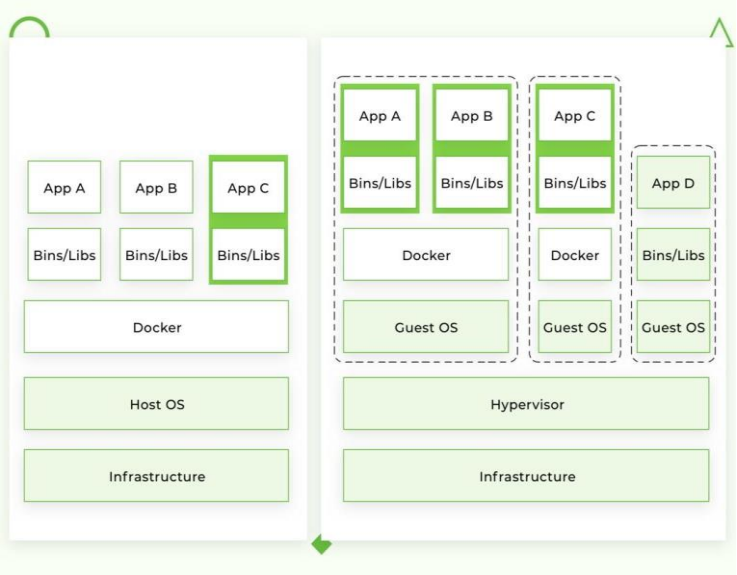


Рисунок 4.3 - Архітектура Docker

4.2 Рекомендації що до створення інформаційної системи

Щоб зрозуміти процес моделювання аналітичної підсистеми треба створити UseCase діаграму (варіантів використання) в якій буде описана модель взаємодії серверу, користувача та адміністратора. Потім треба створити контекстну діаграму IDEF0.

На діаграмі IDEF0(рис 4.5) зображено створення статистичних даних, а внизу механізми, які впливають на процес.

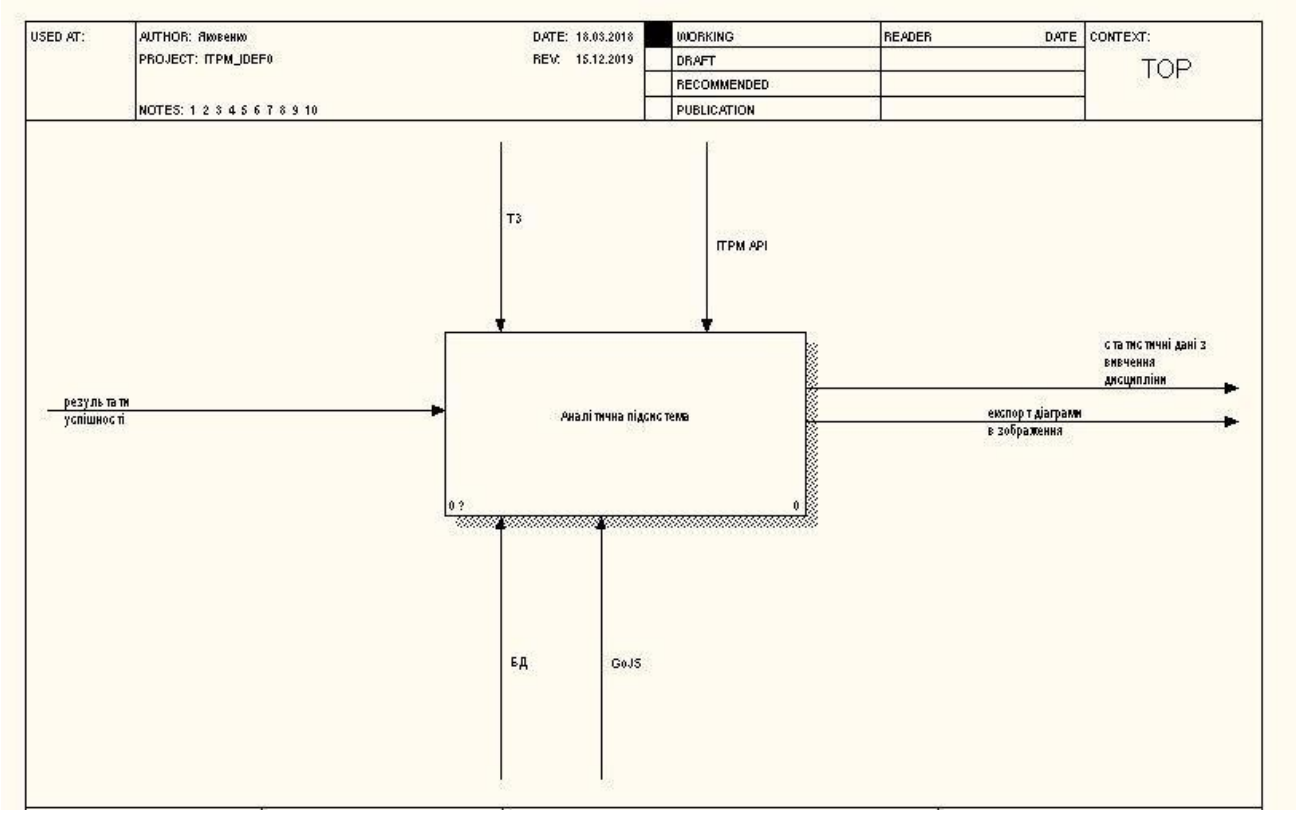


Рисунок 4.5 - IDEF0. Контекстна діаграма аналітичної підсистеми.

Наступним кроком йде декомпозиція контекстної діаграми(рис 4.6). Для статистики є створення своїх діаграм у тесті. При створенні діаграми створюються унікальні ідентифікатори (UUID) за допомогою яких відрізняються користувачі в БД. За процеси, які відносяться до відображення

діаграми в статистиці відповідає «GoJS» . Для процесу «Перегляд статистики» відповідає механізм «Сервер».

Результатом процесів перегляду статистики є діаграма або зображення, яке можна вивантажити. Статистичні дані записуються в БД, а при записі створюють події за допомогою яких буде оновлюватися (рис 4.7).

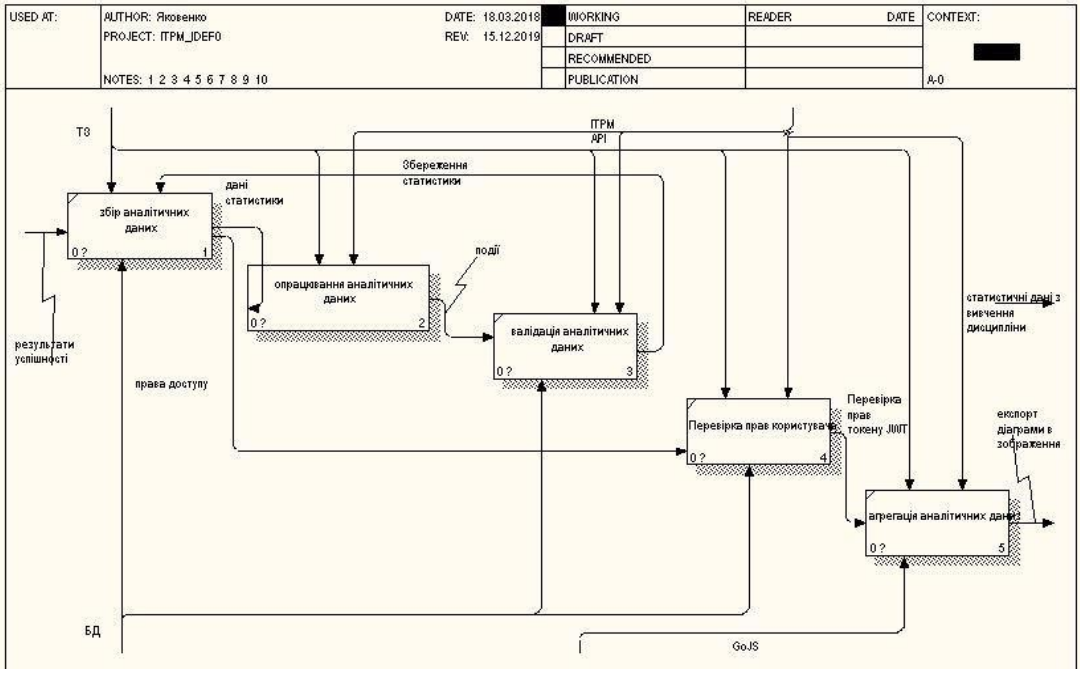


Рисунок 4.6 - Перший рівень декомпозиціїпроцесів аналітичної підсистеми

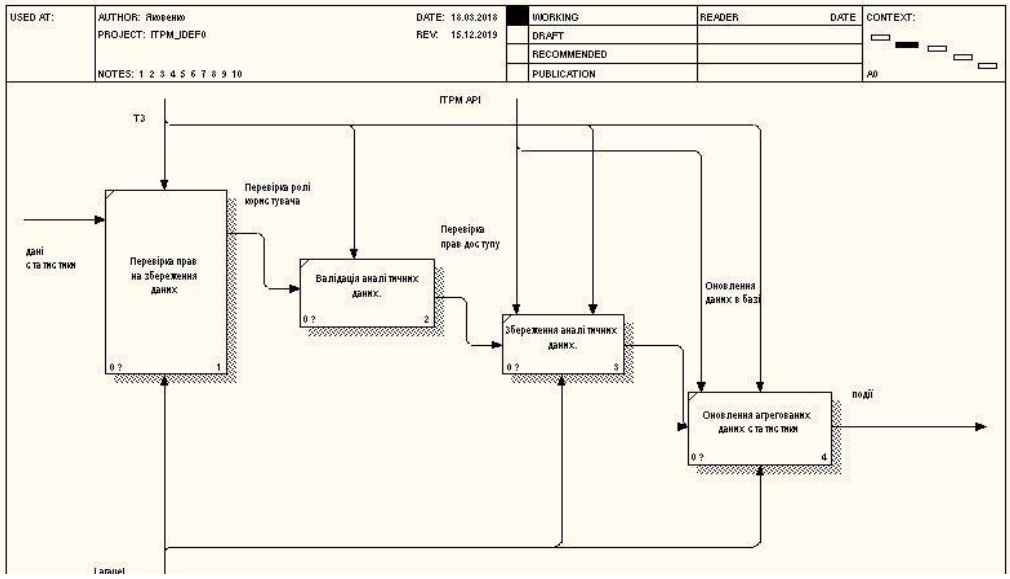


Рисунок 4.7 - Декомпозиція процесу збору аналітичних даних

Створення діаграми варіантів використання. Виділимо 3х «героїв»: сервер, статистика та користувач. Виділимо загальні варіанти використання між користувачем та статистикою, а саме:

Варіанти використання для користувача:

- розмежування прав між адміністратором та користувачем;
- розмежування прав між користувачами;
- перегляд аналітики користувача по діаграмі;
- перегляд аналітики статистики користувача по темі.
- перегляд аналітики користувача по тесту;

Варіанти використання для адміністратора:

- розмежування прав між адміністратором та користувачем;
- розмежування прав між користувачами;
- перегляд аналітики користувача по тесту;
- перегляд конкретної аналітики користувача по темі.
- перегляд аналітики по конкретному користувачу; - soft delete аналітики.
- перегляд аналітики користувача по діаграмі;

Варіанти використання для серверу:

- фільтрація аналітики по користувачу;
- посторінкова навігація по аналітики;
- перевірка прав доступу.
- soft delete аналітики;

- видача лістингу аналітики;
- фільтрація аналітики по статусу виконання;
- фільтрація аналітики по користувачу;

У результаті була створена діаграма варіантів використання(рис 4.8).

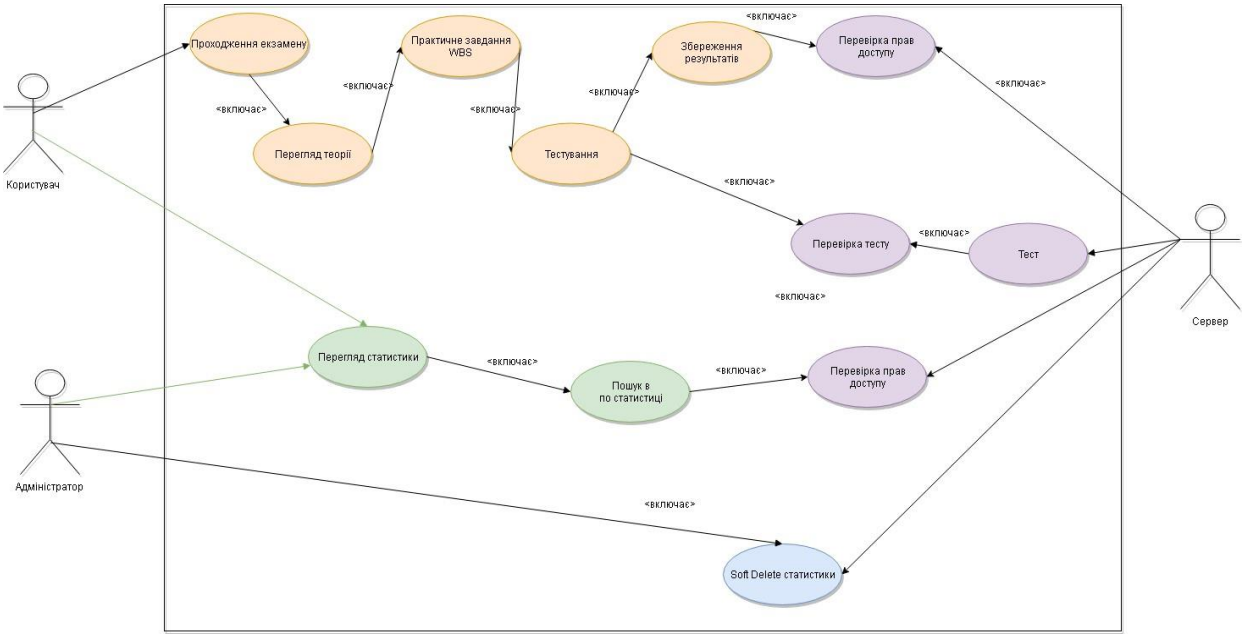


Рисунок 4.8 - Діаграма варіантів використання

4.3 Проектування бази даних

Для зберігання данних з аналітики треба спроектувати дві БД. Перша буде використовуватися для зберігання даних запису, а інша потрібна для зчитування даних та віддачі користувачу.

В результаті моделювання БД для операції запису потрібні такі таблиці:

 статистичні агрегати.

Для моделювання БД операцій зчитування потрібні такі таблиці:

 результати діаграми;

 результати тесту;

- користувачі;
- таблиця міграцій.

При збереженні результату статистики записуємо дані в БД для необроблений даних, які потім будуть агреговані для зчитування.

Таблиця для статистичних агрегатів містить в собі поля для ідентифікації користувача, діаграми, тесту та зберігає результати його і діаграми в jsonb форматі. Таблиці будуть агреговані та зібрані на рівні додатку і записані в окрему базу для зчитування.

Архітектурний шаблон проектування який винує агрегацію даних та операція запису та зчитування називається EventSourcing.

EventSourcing – це збереження в доменній області поточного стану даних, що використовується як інкрементне сховище для запису повних серій дій з даними. Сховище працює як система запису, і його можна використовувати, щоб матеріалізувати об'єкти домену. Це дозволяє полегшити завдання в складних доменах, усуваючи необхідність синхронізації моделі даних і домену для бізнесу при одночасному підвищенні результативності. Також, забезпечується сумісність транзакційних даних і збереження всіх журналів аудиту та історії, за допомогою яких можна використовувати компенсуючі дії.

Більшість додатків працюють з даними, найтиповійший підхід - підтримка поточного стану даних за рахунок оновлення по можливості роботи з ними. Наприклад, у традиційній моделі CRUD (створення, читання, оновлення та видалення) типова обробка даних являє собою зчитування даних зі сховища, внесення певних змін і оновлення їх поточного стану за допомогою нових значень.

Шаблон джерела подій визначає підхід до обробки операцій з даними на основі послідовності подій, кожне з них записується в сховище. Алгоритм додатку відправляє ряд подій, які примусово описують кожну дію з даними в сховище подій. Кожна подія означає деякий набір змін в даних.

Події зберігаються в сховищі подій, яке виступає в якості системи запису поточного стану. Сховище подій зазвичай публікує ці події для оповіщення споживачів і при необхідності надання їм можливості обробки цих події Зверніть увагу, що алгоритм програми, який створює події, ніяк не пов'язаний з системами, підписаними на ці події.

Стандартне використання події, опублікованих сховищем подій, включає підтримку матеріалізованих уявлень сутностей в міру їх зміни в додатку за допомогою дій, а також інтеграцію з зовнішніми системами. Наприклад, система може підтримувати матеріалізоване уявлення всіх замовлень клієнта, використовуваних для заповнення частин визначеного для користувача інтерфейсу. З додаванням нових замовлень, додаванням або видаленням позицій в замовленні або додавання деталей про доставку в додатку події, що описують ці зміни.

Крім того, додатки можуть в будь-який момент зчитати історію подій і використовувати її для матеріалізації поточного стану сутності за рахунок відтворення і використання всіх подій, які відносяться до цієї сутності. Це можна зробити за запитом для матеріалізації об'єкта домену при обробці запиту, в запланованої задачі для збереження стану сутності у вигляді матеріалізованого уявлення для підтримки рівня уявлення.

Нижче представлений огляд шаблону, з деякими варіантами використання потоку подій, наприклад створення матеріалізованого уявлення, інтеграцію подій із зовнішніми програмами і системами, а також відтворення подій за для створення проекцій поточного стану певних сутностей.

Клієнтом та сервер взаэмодыють за допомогою http запитів. Клієнт запитує (певний ресурс), а сервер відповідає результатом по певному ресурсу.

Клієнт може створити, видалити, прочитати чи оновити ресурс якщо в нього є права доступу. Бекенд має розмежування маршрутів по префіксам. За

префіксами маршруту приховується система розмежування прав мыж користуваччами.

Відповідає бекенд клієнту в певному зазначеному стандарті JSON(рис 4.9). Бекенд повинен дотримуватися архітектурних обмежень(рис 4.10). Клієнт відправляє запити на бекенд в зазначену форматі, а бекенд має в цьому форматі відправляти дані назад .

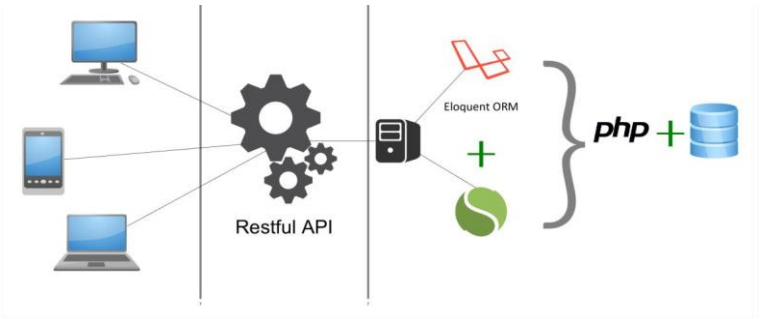


Рисунок 4.9 - Архітектура RESTful бекенду

Проектування структури проекту - проект має буди робитий на чотири частини: фронтенд частина для адміністратору, фронтенд частина для користувача, загальна бібліотека та бекенд частина.

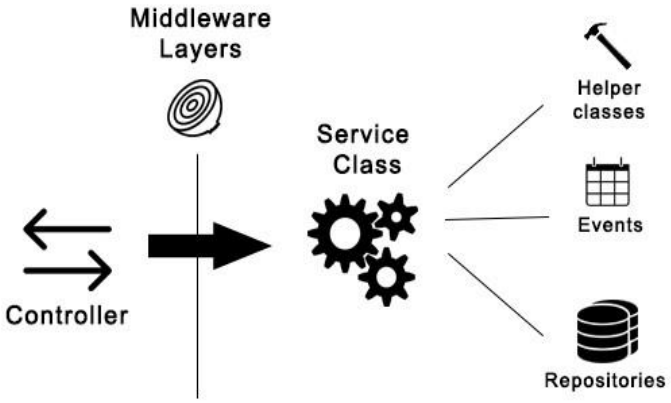


Рисунок 4.10 - Архітектура взаємодії всередині бекенду .

Клієнт відправляє запит по визначеному маршруту, маршрут викликає Middeware – проміжне програмне забезпечення, що перевірє доступ користувача до маршруту. Наступним кроком викликається контролер, який виклиє сервіси

обробки запиту. Сервіси використовують репозиторії для роботи з БД. Для створення події при записі статистичних даних використовується шина даних, яка публікує події підписникам, а підписники обробляють подію.

Алгоритм який виконує проміжні результати і допомагає обробити запит користувача, не містить в собі бізнес логіки по зміні даних - «Helper» або помічник.

Архітектура загальної БД , що містить в собі таблиці для статистики, які були агреговані з таблиці агрегатів необроблених даних результатів тестів.

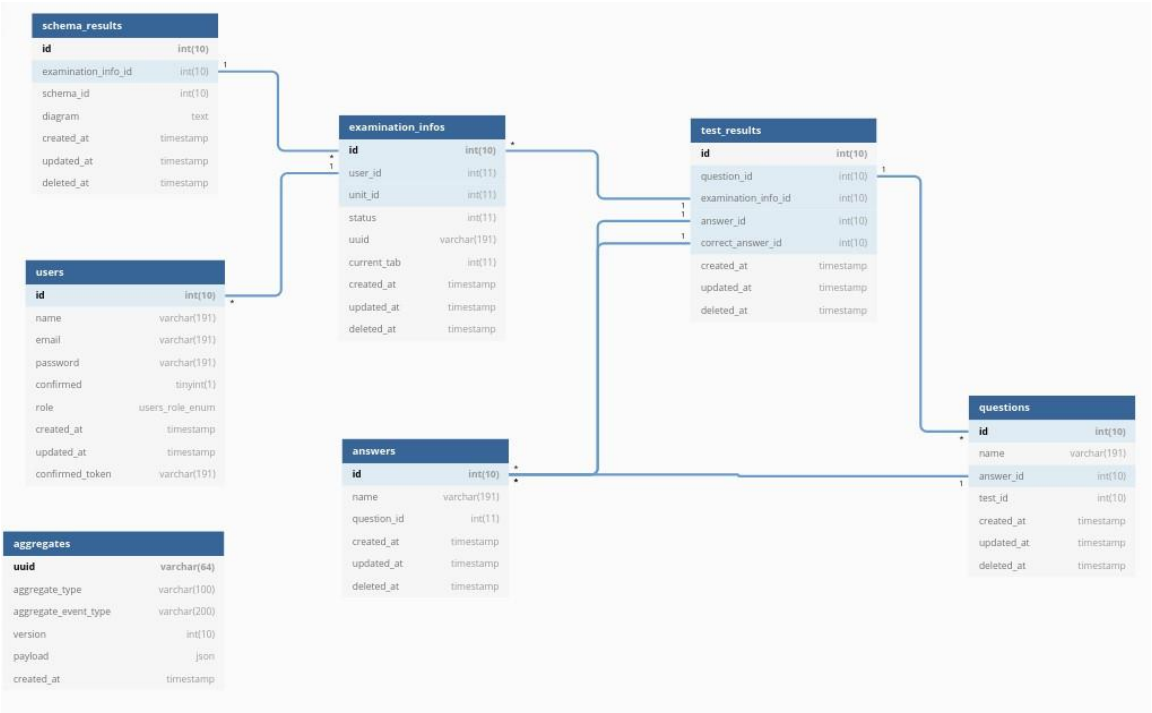


Рисунок 4.11 - Архітектура БД

Архітектура БД(рис 4.11) містить таблицю для міграцій, також таблицю користувачів, курсів, тем, діаграм, тестів,

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даній частиній проводиться оцінка відповідних функціональних та вартісних характеристик продукту, рекомендованого до розробки інструменту менеджера проектів.

Нижче приведеній аналіз різноманітних реалізацій модулю з метою вибору оптимального шляху створення продукту, враховуючи при цьому економічні фактори, характеристики продукту, що впливають на продуктивність роботи і на його сумісність з різноманітним апаратним забезпеченням. Для цього було використано маханізм або метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз(ФВА) – технологія оцінки кінцевої вартості продукту, та (або) послуги незалежно від структури компанії. Прямі та побічні витрати розподіляються в залежності від потрібних обсягів ресурсів на етапах виробництва. Виконані на кожному єтапі дії у контексті метода вартісного аналізу називаються - функціями.

Мета методу полягає у оптимізації розподілу ресурсів, що виділені на продук або послуг у, на прямі та непрямі витрати. У даному випадку – аналіз функцій продукту й виявлення загальних витрат на реалізацію.

Також, він починається з визначення послідовності функцій, необхідних для виробництва. Спочатку йдуть всі можливі функції, які розподіляються по групам: вплив на вартість продукту і ті, що зовсім не роблять ніякого впливу. Також на цьому етапі проводиться оптимізація самої послідовності скороченням кроків, що не впливають на витрати.

Для кожної функції є повний обсяг річних витрат та робочих часів. На основі цих оцінок визначається кількісна характеристика джерел витрат. Після визначення джерел витрат проводиться остаточний розрахунок витрат на виробництво.

5.1 Визначення задач для техніко-економічного аналізу

Метод Функціонально-вартісний аналізу був застосований для проведення техніко-економічного аналізу розробки системи і рекомендованого до розробки інструменту менеджера проектів. Оскільки основні проектні рішення стосуються всієї системи, фактичний аналіз представляє собою визначення функцій продукту, системи керування проектами.

До продукту були визначені наступні технічні вимоги:

1) Застосування на персональних комп’ютерах із стандартною конфігурацією за допомогою веб-браузера;
2) висока швидкість обробки даних;
3) простота взаємодії;
4) мінімізація витрат на впровадження програмного продукту.

5.1.1Обґрунтування функцій.

Головна функція F_0 – розробка алгоритму продукту, який вирішує задачу керування проектами. Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

F_1 – вибір мови програмування;

F_2 – вибір бібліотек для машинного навчання;

F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

1) Python
2) JavaScripy

Функція F_2:

1) NLP.js
2) Natural.js

3) Tensorflow

Функція F_3:

1) PyCharm CE
2) WebStorm
3) Visual Studio

5.1.2Варіанти реалізації функцій

Шляхи реалізацій функцій наведені у морфологічній карті(рис 5.1) системи. Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

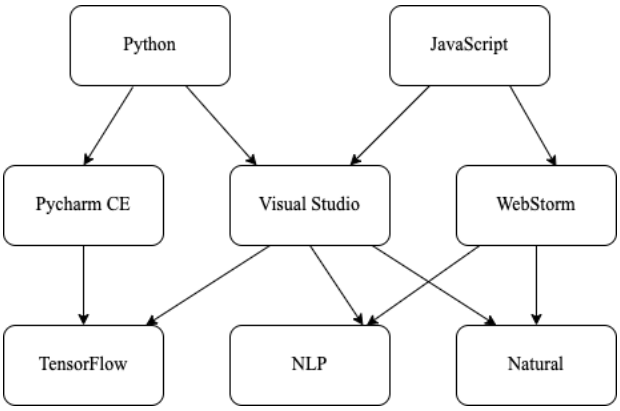


Рисунок 5.1 - Морфологічна карта варіанів реалізації функцій

На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 13).

Таблиця 13 – Позитивно-негативна матриця варіантів основних функцій

Функція	Варіант реалізації	Переваги	Недоліки
F_1	A	Кросплатформений, займає менше часу на виконання коду	Займає більше часу на написання коду

	Б	Займає менше часу на написання коду, кросплатформений, може бути використаний для будь-яких типів додатків	Займає більше часу для виконання коду
F_2	А	Безкоштовність, добра документація, постійно оновлюється	Відсутня реалізація додатку для Windows
	Б	Безкоштовність, гарна документація, підтримка багатьох мов	Відсутня підтримка
F_3	А	Гарна підтримка, швидкодія	Тільки для Python
	Б	Безкоштовний Підходить як для JS, так і для Python	Необхідність довгих налаштувань для оптимізації роботи
	В	Потребує менше пам’яті, можливо виконувати окремі ділянки коду	Більше підходить для JS, платний

На основі аналізу робимо висновок, що при розробці продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Оскільки реалізація програмного коду для вирішення задачі є концептуально складною, необхідно обрати мову, що спрощує написання коду, тому варіант А має бути відкинутий.

Функція F_2:

Вибір фреймворку не відіграє велику роль у даному продукті, тому вважаємо варіанти Б та В гідними розгляду.

Функція F_3:

Оскільки, продукт реалізується мовою JS, можемо відкинути варіант А.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

- F_1(Б) – F_2(Б) – F_3(Б)
- F_1(Б) – F_2(В) – F_3(Б)

- F_1(Б) – F_2(Б) – F_3(В)
- F_1(Б) – F_2(В) – F_3(В)

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.2 Параметри програмного продукту

5.2.1Опис параметрів

На підставі обраних даних про функції, що повинен мати продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта тех-рівня.

Для того, щоб охарактеризувати продукт, будемо використовувати наступні параметри:

- X1 – швидкодія операцій;
- X2 – об’єм пам’яті для збереження даних, необхідний під час виконання програми;
- X3 – час, який витрачається на виконання;
- X4 – можливий об’єм програмного коду.

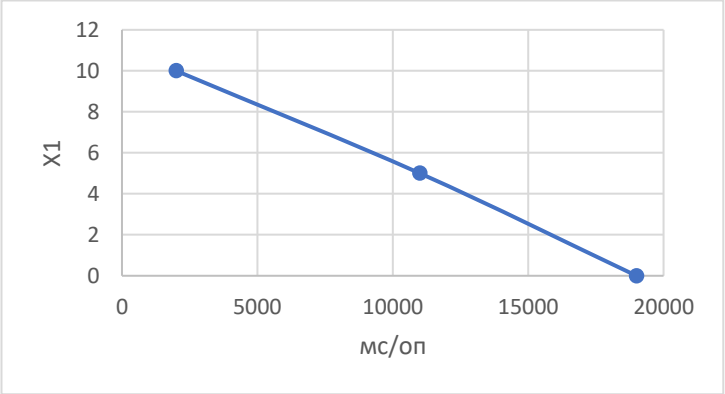
5.2.2 Кількісна оцінка параметрів

Усі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту як показано нижче у таблиці 14.

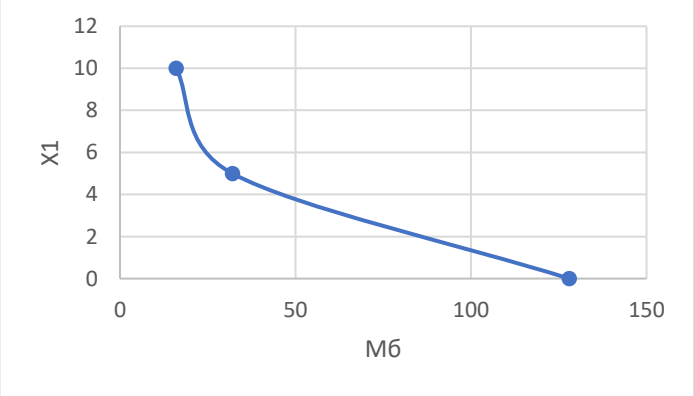
Таблиця 14 Основні параметри програмного продукту

Опис параметру	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	мс/оп	3000	11000	18000
Об’єм пам’яті для збереження даних	X2	Мб	128	64	16
Час виконання коду	X3	с	1000	600	200
Потенційний об’єм програмного коду	X4	строк коду	2000	1500	1000

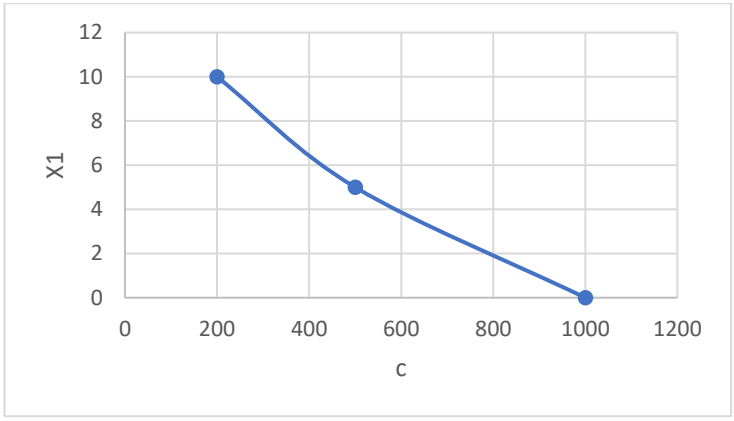
За даними таблиці будуються графічні характеристики параметрів(рис 5.2-5)



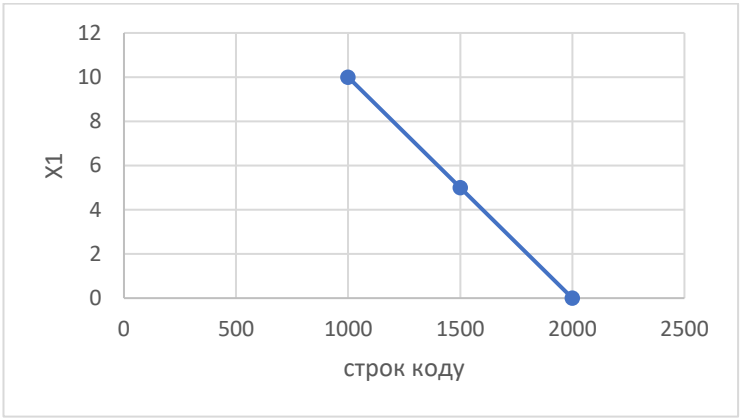
Рисункок 5.2 - Швидкодія



Рисункок 5.3 - Об’єм пам’яті



Рисункок 5.4 - Час виконання



Рисунко 5.5 - Можливий об’єм програмного коду

5.2.3Аналіз експертного оцінювання параметрів

Після детального аналізу кожний експерт визначає ступінь важливості кожного параметру для конкретно поставленої задачі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем

Важливість кожного параметра визначається методом попарного порівняння. Оцінку проводить комісія із 5 людей. Результати експертного ранжування наведені у таблиці 15. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра, або присвоєння різних рангів;

2) перевірку придатності експертних оцінок;

3) визначення оцінки попарного пріоритету параметрів;

4) обробку результатів та визначення коефіцієнту значимості.

Таблиця 15 Результати ранжування показників

Параметр	Ранг параметра за оцінкою експерта					Сума рангів	Відхилення, Δ_i	Δ_i^2
	1	2	3	4	5			
X1	3	4	4	3	4	18	+5,5	30,25
X2	2	1	1	2	2	8	–4,5	20,25
X3	1	2	2	1	1	7	–5,5	30,25
X4	4	3	3	4	3	17	+4,5	20,25
Разом	10	10	10	10	10	50	0	101

Для перевірки ступеню достовірності оцінок, визначимо наступні параметри:

а) сума і загальна сума рангів кожного з параметрів:

$$R_i = \sum_{j=1}^{N} * r_{ij} = 50,$$

де r_{ij} – ранг i-го параметра, визначений j-м експертом;
N – число експертів.

б) середня сума рангів T:

$$T = \frac{1}{n} R_i = 12{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{n} * \Delta_i^2 = 101.$$

д) коефіцієнт узгодженості (конкордації):

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 101}{5^2(4^3 - 4)} = 0{,}808 > W_k = 0{,}67.$$

Ранжирування вважаємо достовірним, бо знайдений коефіцієнт узгодженості перевищує нормативний(0,67). Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів, результати заносимо у таблицю

16. Числове значення, що визначає ступінь переваги i-го параметра над j-тим, a_{ij} визначається за формулою:

$$a_{ij} = \{1{,}5 \;\; x_i > x_j;\; 1{,}0 \;\; x_i = x_j;\; 0{,}5 \;\; x_i < x_j.$$

Таблиця 16 Результати ранжування параметрів

Параметри	Експерти					Підсумкова оцінка	Числове значення коефіцієнтів переваги
	1	2	3	4	5		
X1,X2	>	>	>	>	>	>	1,5
X1,X3	>	>	>	>	>	>	1,5
X1,X4	<	>	>	<	>	>	1,5
X2,X3	>	<	<	>	>	>	1,5
X2,X4	<	<	<	<	<	<	0,5
X3,X4	<	<	<	<	<	<	0,5

З отриманих числових оцінок складемо матрицю $A = \|a_{ij}\|$. Для кожного параметра розрахунок вагомості K_{B_i} проводиться за формулою:

$$K_{B_i} = \frac{b_i}{\sum_{i=1}^{n} * b_i},$$

де $b_i = \sum_{j=1}^{N} a_{ij}$ – вагомість i-го параметра за результатами оцінок всіх експертів;

a_{ij} –коефіцієнт переваги i-го на j-тим параметром.

Відносні оцінки розраховуються декілька разів, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступною формулою:

$$K_{B_i} = \frac{b_i'}{\sum_{i=1}^{n} * b_i'},$$

де $b_i' = \sum_{j=1}^{N} * a_{ij} b_j$.

Як видно з наступної таблиці 17, різниця значень коефіцієнтів вагомості після другої ітерації не перевищує 2%, тому додаткові ітерації не потрібні.

Таблиця 17 Розрахунок вагомості параметрів

i	j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	B_i	K_{B_i}	B_i^1	$K_{B_i}^1$

X1	1,0	1,5	1,5	1,5	5,5	0,344	21,25	0,36
X2	0,5	1,0	1,5	0,5	3,5	0,219	12,25	0,208
X3	0,5	0,5	1,0	0,5	2,5	0,156	9,25	0,157
X4	0,5	1,5	1,5	1,0	4,5	0,281	16,25	0,275
Разом					16,0	1,0	59,0	1,0

5.3 Аналіз якості різних реалізацій функцій

Рівень якості кожного варіанту виконання функцій визначається окремо.

Абсолютні значення параметрів X1(швидкодія) та X2 (об'єм пам'яті) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час виконання) обрано не максимальним, тобто це значення відповідає варіантам б) 500 та в) 200 с.

Абсолютне значення параметра X4 (кількість коду) обрано також не найгіршим(за к-тю строк), тобто це значення відповідає варіанту б) 1500 та в) 1000.

Коефіцієнт технічного рівня якості для кожного варіанта реалізації ПП розраховується за формулою:

$$K_{TP} = \sum_{i=1}^{n} * K_{B_i} B_i,$$

де n – кількість параметрів;
K_{B_i} – коефіцієнт вагомості і–го параметра;
B_i – оцінка i–го параметра в балах.

Розрахунок показників рівня якості представлено відповідно в таблиці 18.

Таблиця 18 Розрахунок показників якості

Основні функції	Варіант реалізації	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А	32	5	0,208	1,16
F_2	А	11000	5	0,36	1,9

F_3	А	1000	10	0,275	2,37
	Б	1500	5	0,275	1,375
F_4	А	200	10	0,657	1,784
	Б	500	5	0,157	0,478

За цими даними визначаємо рівень якості кожного з варіантів:

– $F_1(А) – F_2(А) – F_3(А) – F_4(А) = 1,16+ 1,9 + 2,37+ 1,784= 7,214$

– $F_1(А) – F_2(А) – F_3(Б) – F_4(Б) = 1,16+ 1,9 + 1,375+ 0,478= 4,913$

– $F_1(А) – F_2(А) – F_3(А) – F_4(Б) = 1,16+ 1,9 + 2,37+ 0,478= 5,908$

– $F_1(А) – F_2(А) – F_3(Б) – F_4(А) = 1,16+ 1,9 + 1,375+ 1,784= 6,219$

Тому, найкращим є перший варіант, для котрого коефіцієнт технічного рівня має найбільше значення.

5.4 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки продукту треба провести розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка алгоритму збору даних.

Завдання 1 за новітністю відноситься до групи А, завдання 2 – до групи Б. За складність, 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовує інформацію у вигляді даних, а завдання 2 використовує методи збору даних. Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}$$

де $Т_Р$ – трудомісткість розробки програмного продукту;
$К_П$ – поправочний коефіцієнт;
$К_{СК}$ – коефіцієнт на складність вхідної інформації;
$К_М$ – коефіцієнт рівня мови програмування;
$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних

програм;
$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для 1 завдання, трудомісткість дорівнює: $Т_Р = 46$ людино-днів. Корегуючий коефіцієнт, який враховує вид вхідної інформації для першого завдання: $К_П = 1,7$. Корегуючий коефіцієнт, який враховує складність контролю вхідної та вихідної інформації рівний $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, вирахуємо це за допомогою коефіцієнта $К_{СТ} = 0,8$. Отже, загальна трудомісткість кодування першого завдання дорівнює:

$$Т_1 = 46 \cdot 1.7 \cdot 0.8 = 62,56 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для завдання 2, в якому використовується алгоритм 3-ї групи складності з новизною Б, тобто $Т_Р = 46$ людино-днів, $К_П = 0,9$, $К_{СК} = 1$, $К_{СТ} = 0,8$:

$$Т_2 = 15 \cdot 0.9 \cdot 0.8 = 10,8 \text{ людино-днів.}$$

Оскільки загальна трудомісткість усіх реалізацій збігається, їх можна об’єднати в одну групу.

Загальна трудомісткість складає:

$$Т_0 = (62,56 + 10,8) \cdot 8 = 586,88 \text{ людино-годин;}$$

В розробці беруть участь програміст з заробітньою платою 17000 грн., один аналітик в області даних -15000 грн -, та один інженер даних з окладом 18000 грн. Середня зарплата за годину визначається за формулою:

$$С_Ч = \frac{M}{T_m \cdot t} \text{ грн.,}$$

де M – місячний оклад працівників;
T_m – кількість робочих днів на місяць;
t – кількість робочих годин в день.

$$С_Ч = \frac{17000 + 15000 + 18000}{3 \cdot 20 \cdot 8} = 104,16 \text{ грн.}$$

Тоді, розрахуємо зарплату за формулою:

$$С_{ЗП} = С_Ч \cdot T_i \cdot К_Д,$$

де $С_Ч$ – величина погодинної оплати праці програміста;
T_i – трудомісткість відповідного завдання;
$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$С_{ЗП} = 104{,}16 \cdot 586{,}88 \cdot 1{,}6 = 97\,807 \text{ грн.}$$

Відрахування на соціальний внесок становить 22%:

$$С_{СВ} = С_{ЗП} \cdot 0{,}22 = 97\,807 \cdot 0{,}22 = 21\,517{,}55 \text{ грн.}$$

Визначимо витрати на оплату однієї машино-години. Оскільки одна машина обслуговується одним інженером апаратного забезпечення за 18000 грн. та коефіцієнтом зайнятості $К_З = 0{,}2$ то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 18\,000 \cdot 0{,}2 = 43\,200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 43\,200 \cdot (1 + 0{,}2) = 51\,840 \text{ грн.}$$

Відрахування на соціальний внесок становить 22%:

$$С_{СВ} = С_{ЗП} \cdot 0{,}22 = 51\,840 \cdot 0{,}22 = 11\,404{,}8 \text{ грн.}$$

Амортизаційні витрати розраховуємо за формулою при амортизації 25% та вартості ЕОМ – 75 000 грн.:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}25 \cdot 37000 = 10\,637{,}5 \text{ грн.}$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;
$К_А$ – річна норма амортизації;
$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт або профілактику розраховуємо за формулою:

$$С_Р = К_{ТМ} \cdot К_Р \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}05 \cdot 37000 = 2\,127 \text{ грн.}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t \cdot К_В, Т_{ЕФ} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0{,}9 = 1\,667{,}6 \text{ год.}$$

де $Д_К$ – календарна кількість днів у році;
$Д_В, Д_С$ – відповідно кількість вихідних та святкових днів;
$Д_Р$ – кількість днів планових ремонтів устаткування;
t – кількість робочих годин в день;
$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_C \cdot К_З \cdot Ц_{ЕЛ} = 1\,677{,}6 \cdot 1{,}2 \cdot 0{,}2 \cdot 0{,}34564 = 139{,}16 \text{ грн.}$$

де N_C – середньо-споживча потужність приладу;
$К_З$ – коефіцієнтом зайнятості приладу;
$Ц_{ЕЛ}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0{,}67 = 37\,000 \cdot 0{,}67 = 24790 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть складати:

$$С_{ЕК} = С_{ЗП} + С_{СВ} + С_А + С_Р + С_{ЕЛ} + С_Н, С_{ЕК} = 21\,517{,}55 + 11\,404{,}8 + 10\,637{,}5 + 2\,127 + 139{,}16 + 24790 = 70616{,}01 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{МГ} = \frac{С_{ЕК}}{Т_{ЕФ}} = \frac{70616{,}01}{1\,677{,}6} = 42{,}09 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$С_М = С_{МГ} \cdot Т = 42{,}09 \cdot 996{,}64 = 41\,952{,}40 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0{,}67 = 97\,807 \cdot 0{,}67 = 65\,530{,}69 \text{ грн.}$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{СВ} + С_М + С_Н, С_{ПП} = 97\,807 + 11\,404{,}8 + 41\,952{,}40 + 24\,790 = 175\,954{,}2 \text{ грн.}$$

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР} = \frac{К_К}{С_{ПП}} = \frac{7{,}214}{175\,954{,}2} = 4{,}10 \cdot 10^{-5}$$

5.5 Висновок

Виконавши розрахунки економічного розділу та провівши аналіз рентабельності, були систематизовані і закріплені теоретичні знання в галузі економіки та організації виробництва використанням їх для техніко-економічного обґрунтування розробки.

На основі даних про зміст функцій, які повинен реалізовувати продукт, були визначені чотири найперспективніші варіанти реалізації продукту. Найефективнішим виявився 3 варіант реалізації функцій ПП, який дає максимальну величину коефіцієнта техніко-економічного рівня, вартість витрат для нього становить $С_{ПП} = 175\,954{,}2$ грн.

Детальні рекомендації що до створення продукту знаходяться у розділі вище.

ВИСНОВОК

В межах цієї дмпломної роботи було проаналізовано та порівняно, які методи контролю і планування операцій здійснюються в системах. Було порівняно обидва традиційні та сучасні методи та розглянуто їх переваги та недоліки. Проведений аналіз систем управління вартістю і користю функціоналу в умовах безлічі користувачів.

Продуктом проекту є рекомендація по створенню аналітичноі підсистема веб-орієнтованого середовища управління it проектами. В результаті роботи було теоретично спроектовано можливості: лістингу статистики з фільтрацією по користувачу, по імені користувача або темі, по статусу проходження, реалізовано видалення статистики, що дозволяє користувачу проходити тест повторно; адміністратор повинен мати можливість переглянути всю статистику по користувачу. Для адміністратора рекомендується реалізувати список активних користувачів, які проходять тест по конкретній темі, розмежування прав перегляду статистики: користувач може переглядати тільки свою статистику, а адміністратор статистику кожного користувача.

ДЖЕРЕЛА

1. Agile Development at Scale: The Next Frontier. [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/331448369_Agile_Development_at_Scale_The_Next_Frontier

2. Key Lessons From Tailoring Agile Methods for Large-Scale Software Development [Електронний ресурс] – Режим доступу до ресурсу: (PDF) Key Lessons From Tailoring Agile Methods for Large-Scale Software Development (researchgate.net)

3. Agile manifesto [Електронний ресурс] – Режим доступу до ресурсу: http://agilemanifesto.org.

4. Henrik Kniberg. Scrum and XP from the trenches, 2007. – 140 с.

5. Scrum [Електронний ресурс] – Режим доступу до ресурсу: http://www.mountaingoatsoftware.com/scrum.

6. Extreme programming [Електронний ресурс] – Режим доступу до ресурсу: http://www.xprogramming.com/xpmag/whatisxp.htm.

7. Kanban [Електронний ресурс] – Режим доступу до ресурсу: https://www.agilealliance.org/kanban.

8. What is Agile Kanban Methodology? [Електронний ресурс] – Режим доступу до ресурсу: https://www.inflectra.com/Methodologies/Agile-Development.aspx.

9. SDLC - Waterfall Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/waterfall_model.

10. Classical Waterfall Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/.

11. V-Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm.

12. Software Engineering. SDLC V-Model. [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/software-engineering-sdlc-vmodel

13. V-Model: An Improvement of Waterfall. [Електронний ресурс]– Режим доступу до ресурсу: https://medium.com/software-engineering-kmitl

14. V-Model in Software Testing[Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/v-model-software-testing.html

15. Iterative Model: What Is It And When Should You Use It? [Електронний ресурс] – Режим доступу до ресурсу: https://airbrake.io/blog/sdlc/iterative-model

16. SDLC. Iterative Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

17. SDLC. Spiral Model. [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

18. What Is SDLC Spiral Model? [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/spiral-model-what-is-sdlc-spiral-model/

19. Clarizen - повнофункціональне програмне забезпечення для управління проектами [Електронний ресурс] - Режим доступу до ресурсу : https://www.clarizen.com/

20. monday.com - інструмент планування з дошками kanban, трекер проектів [Електронний ресурс] – Режим доступу до ресурсу: https://monday.com

21. Celoxis - веб-інструмент із всеосяжними функціями управління проектами та портфелем [Електронний ресурс] - Режим доступу до ресурсу: https://celoxis.com.

22. 10,000ft - програмне забезпечення для управління проектами [Електронний ресурс] - Режим доступу до ресурсу: https://www.10000ft.com.

23. Ravetree - нагородами рішення для управління проектами з великою кількістю вбудованих функцій спритного управління проектами [Електронний ресурс] - Режим доступу до ресурсу: https://www.ravetree.com.

24. Wrike - програмне забезпечення для співпраці та управління проектами на основі хмар, яке легко масштабувати [Електронний ресурс] – Режим доступу до ресурсу: https://www.wrike.com.

25. Angular – фреймворк для фронтенду розроблений компанією Google [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/.

26. Javascript - мова програмування бекенду, фронтенду [Електронний ресурс] Режим доступу до ресурсу: https://learn.javascript.ru/intro

27. TypeScript — мова програмування для розширення можливостей JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/.

28. PHP — мова програмування, яка дозволяє за допомогою скриптів створювати бекенд [Електронний ресурс] – Режим доступу до ресурсу: http://php.net/.

29. Laravel — PHP фреймворк, який використовується для написання бекенду [Електронний ресурс] — Режим доступу до ресурсу: https://laravel.com.

30. MySQL —система управління базами даних [Електронний ресурс] — Режим доступу до ресурсу: https://www.mysql.com/.

31. RxJS — бібліотека для реактивного програмування [Електронний ресурс] Режим доступу до ресурсу: http://reactivex.io/rxjs/manual/index.html.

32. Docker — система віртуалізації на основі механізмів Linux [Електронний ресурс] - Режим доступу до ресурсу: https://www.docker.com/.

33. GoJS — бібліотека для створення інтерактивних діаграм [Електронний ресурс] - Режим доступу до ресурсу: https://gojs.net/latest/index.html.

34. DHTMLX — бібліотека для створення діаграмГанта [Електронний ресурс] - Режим доступу до ресурсу: https://dhtmlx.com/.

35. Google Cloud — «хмарна» платформадля підняття інфраструктури [Електронний ресурс] - Режим доступу до ресурсу: https://cloud.google.com/.

36. Github — сервіс для надання хостингу коду [Електронний ресурс] - Режим доступу до ресурсу: https://github.com/.

37. CQRS – архітектурний шаблон проектування [Електронний ресурс] — Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/azure/architecture/patterns/cqrs

38. Роберт Седжвик. Алгоритми на C++. Boston/San Francisco/NewYork/Toronto/Montreal/London/Munich/Paris/Madrid/Capetown/Tokyo/Singapore/Mexico City — Addison-Wesley, 2014 —634-645.

39. EventSourcing – архітектурний шаблон проектування [Електронний ресурс] — Режим доступу до ресурсу: https://martinfowler.com/eaaDev/EventSourcing.html