Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи магістра

на тему: «Аналіз процедур збору та обробки даних життєвого циклу розробки програмного забезпечення»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ2322»

_____ /Георгій СІНЬКОВ/
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Вадим ГОРЯЧКІН/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Світлана ВОЛКОВА/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

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

Дніпро – 2025 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Master's Thesis

on the topic: «Analysis of software development life cycle data collection and processing procedures»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1911: /Heorhii SINKOV/

Scientific Supervisor: /Vadym HORIACHKIN/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2025

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: Факультет «Комп'ютерні технології і системи»

Кафедра: «Комп'ютерні інформаційні технології»

Рівень вищої освіти: магістр

Освітня програма: «Інженерія програмного забезпечення»

Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

_____ /Вадим ГОРЯЧКІН/

(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу Магістр

студенту Сінькову Георгію Олексійовичу

1. Тема роботи: «Аналіз процедур збору та обробки даних життєвого циклу розробки програмного забезпечення»

Керівник роботи: Горячкін Вадим Миколайович, завідувач кафедрою, к.т.н., доцент

затверджені наказом № 1209 ст від 00.00.2025

2. Строк подання студентом роботи: 00.00.2025р.

3. Вихідні дані до роботи: Для виконання цієї роботи використовувались різноманітні джерела інформації, зокрема, теоретичні матеріали про сучасні методології розробки програмного забезпечення, такі як Scrum, Kanban та Waterfall. Також враховувались дані про метрики, що використовуються для оцінки продуктивності та якості роботи команд розробників. Окрім цього,

були розроблені та використані моделі для симуляції процесів розробки програмного забезпечення, що дозволило більш детально проаналізувати ефективність різних підходів. Важливим джерелом даних стали результати дослідження впливу розміру команди на продуктивність проектів, що дало можливість оцінити цей фактор у контексті різних методологій розробки.

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

Вступ: актуальність теми, мета, завдання та методи дослідження.

Розділ 1: аналіз сучасних методологій розробки програмного забезпечення.

- Опис основних методологій (Scrum, Kanban, Waterfall).
- Переваги та недоліки кожної методології.
- Метрики для оцінки продуктивності команд.

Розділ 2: Моделювання процесів розробки програмного забезпечення.

- Створення імітаційних моделей для обраних методологій.
- Логіка роботи моделей і використання метрик.

Розділ 3: Аналіз результатів застосування імітаційних моделей.

- Оцінка ефективності кожної методології на основі симуляцій.
- Порівняння результатів та висновки щодо оптимального вибору методології для конкретних умов.

Розділ 4: Дослідження впливу розміру команди на продуктивність у різних методологіях.

- Аналіз залежності продуктивності від кількості учасників команди.

Висновки: підсумок основних результатів дослідження.

Рекомендації щодо покращення процесів збору та обробки даних у розробці ПЗ.

5. Перелік графічного матеріалу (з точним зазначанням обовязкових креслень):

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	04.11.24 – 10.11.24	
2	Постановка задачі, технічне завдання	04.11.24 – 10.11.24	30%
3	Розробка інструментальних засобів дослідження	09.12.24 – 15.12.24	
4	Виконання дослідження	09.12.24 – 15.12.24	60%
5	Оформлення тез доповідей	06.01.25 – 12.01.25	
6	Оформлення пояснювальної записки	06.01.25 – 12.01.25	
7	Разробка демонстаційних матеріалів	06.01.25 – 12.01.25	100%
8	Подання кваліфікаційних роботи до кафедри	09.01.25	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної роботи	23.01.25	

Студент _____ Георгій СІНЬКОВ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Вадим ГОРЯЧКІН
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 8 розділів:

Об’єктом дослідження є життєвий цикл розробки програмного забезпечення (ЖЦРПЗ), що включає процеси формулювання вимог, проектування, розробки, тестування, впровадження та супроводу програмних продуктів.

Предметом дослідження є процедури збору та обробки даних на різних етапах життєвого циклу розробки програмного забезпечення, включаючи використання сучасних інструментів та метрик для оцінки продуктивності та якості роботи команд розробників.

Метою роботи є дослідження процедур збору та обробки даних у контексті ЖЦРПЗ, оцінка їхньої ефективності та формулювання рекомендацій для їхнього вдосконалення. Зокрема, дослідження зосереджуватиметься на аналізі таких аспектів, як вибір метрик, оптимізація процесів збору даних, автоматизація їх обробки та візуалізація результатів для подальшого використання у прийнятті рішень

Методи дослідження У роботі використовуються моделювання методологій розробки програмного забезпечення (Scrum, Kanban, Waterfall), які дозволяють аналізувати ефективність різних підходів у контексті життєвого циклу розробки. Моделі враховують реальні умови, зокрема, випадкові відхилення у тривалості завдань та ймовірність дефектів, щоб симулювати реалістичні процеси розробки

Результати та їх новизна: виконано аналіз моделей та процедур широкого кола завдань менеджменту програмної інженерії при управлінні проектами. Розроблено та модифіковано моделі Scrum, Kanban і Waterfall для симуляції різних підходів до розробки програмного забезпечення. У процесі дослідження запропоновано нові методики оцінки продуктивності команд розробників, зокрема використання метрик продуктивності та якості для оптимізації робочих процесів. Для аналізу ефективності застосованих моделей

проведено численні симуляції, що дозволили підтвердити достовірність і практичну значущість отриманих результатів. Новизна дослідження полягає у практичному застосуванні моделей, які враховують реалістичні умови розробки програмного забезпечення. Це дозволяє зробити висновки більш обґрунтованими та практично значущими для різних методологій. Результати можуть бути використані для вдосконалення підходів до збору та обробки даних у програмній інженерії, що сприяє підвищенню ефективності командної роботи та поліпшенню якості програмного забезпечення.

Розрахунково-пояснювальна записка складається із 5 розділів, висновків, бібліографічниного списку та додатків.

Вступ – в даному розділі описується сутність розробки, її актуальність. Визначає актуальність завдань досліджень та розробки. (2 сторінок)

Перший розділ – у цьому розділі розглядаються різні методології розробки програмного забезпечення, такі як Scrum, Kanban і Waterfall. Аналізуються їхні особливості, сильні та слабкі сторони, а також їхній вплив на ефективність роботи команд. Особлива увага приділяється метрикам, які використовуються для оцінки продуктивності та якості роботи команд розробників. (24 сторінок)

Другий розділ – цей розділ присвячений створенню імітаційних моделей для обраних методологій розробки. Описуються основні елементи моделей, логіка їх роботи, а також метрики, що використовуються для оцінки результатів. Введено механізми симуляції, які дозволяють оцінити реальні умови виконання завдань у проекті. (11 сторінок)

Третій розділ – у цьому розділі проводиться аналіз результатів моделювання різних методологій розробки програмного забезпечення, таких як Scrum, Kanban і Waterfall. На основі отриманих даних та метрик оцінюється ефективність кожної методології в різних контекстах. Після порівняння результатів визначається оптимальна методологія для застосування у конкретних умовах проекту. (8 сторінок)

Четвертий розділ – у цьому розділі досліджується вплив розміру команди на продуктивність виконання проектів у різних методологіях розробки

програмного забезпечення, таких як Scrum, Kanban і Waterfall. Аналізуються результати числових експериментів, що дозволяють оцінити, як зміна кількості учасників команди впливає на швидкість виконання завдань, якість продукту та загальну ефективність проекту. (8 сторінок)

Висновки. Складається з 1 сторінки. Додатки – технічне завдання і робочий проект. Таблиці – 9, рисунків – 18, бібліографія – 5.

Ключові слова: методологія, модель, моделювання, Scrum, Kanban, Waterfall, метрики.Пояснювальна записка до кваліфікаційної роботи магістра: (рівень освіти) 96с., 18 рис., 6 табл., 4 додатка, 5 джерел

Зміст

Вступ

Життєвий цикл розробки програмного забезпечення (ЖЦРПЗ) становить комплексний процес, що охоплює етапи від формулювання вимог до впровадження та супроводу програмних продуктів. У сучасних умовах швидкого розвитку інформаційних технологій зростає потреба у вдосконаленні методів збору та обробки даних, які супроводжують ці етапи. Ефективне управління даними, отриманими на різних стадіях ЖЦРПЗ, є ключовим фактором для забезпечення якості програмного забезпечення, оптимізації ресурсів та підвищення продуктивності команд розробників.

Збір та обробка даних на етапах життєвого циклу програмного забезпечення має вирішальне значення для підвищення прозорості та контрольованості процесів, що дозволяє командам розробників приймати об'єктивні та обґрунтовані рішення. Сучасні інструменти, такі як Jira, Trello, Azure DevOps та інші, пропонують широкий диапазон метрик для оцінювання продуктивності та якості командної роботи. Впровадження цих методів та їх адаптація до конкретних проектів дозволяють перетворити дані на цінний ресурс для прогнозування та планування.

Разом із тим, аналіз процедур збору та обробки даних висвітлює низку викликів. Серед них можна виділити проблему вибору релевантних метрик, складність інтеграції інструментів у робочий процес, а також забезпечення точності та надійності даних. Ці аспекти є особливо актуальними у великих проектах з розподіленими командами, де координація діяльності стає критичною. Вирішення таких проблем вимагає як теоретичного, так і практичного підходу до вивчення доступних методів та інструментів.

Метою даної магістерської роботи є дослідження процедур збору та обробки даних у контексті ЖЦРПЗ, оцінка їхньої ефективності та формулювання рекомендацій для їхнього вдосконалення. Зокрема, дослідження зосереджуватиметься на аналізі таких аспектів, як вибір метрик, оптимізація процесів збору даних, автоматизація їх обробки та візуалізація

результатів для подальшого використання у прийнятті рішень.

Результати цього дослідження можуть бути використані для розробки нових або вдосконалення існуючих підходів до збору та обробки даних у програмній інженерії. Вони сприятимуть підвищенню ефективності командної роботи, поліпшенню якості програмного забезпечення та зниженню ризиків, пов'язаних з управлінням проектами. Таким чином, це дослідження робить вагомий внесок у розвиток практик управління проектами у сфері програмної інженерії.

Розділ 1 Аналіз та вимоги

У цьому розділі розглядаються моделі, що описують процес створення програмного забезпечення — від початкового планування до завершення проекту. Також аналізуються метрики, які використовуються для оцінки ефективності команд розробників. Важливо зазначити, що аналіз цих моделей і метрик дозволяє виявити як сильні сторони, так і можливі недоліки у процесах розробки. Завдяки цьому можна оптимізувати роботу команд, покращити якість кінцевого продукту та зменшити ризики невдачі проекту.

1.1 Методології розробки програмного забезпечення

1.1.1 Agile (Гнучка методологія)

Гнучкий підхід почав своє існування у першій половині XX століття, хоча є думка, що його елементи були відомі ще раніше. Близько 30-х років фізик Волтер Шухарт впровадив ітеративний підхід Plan-Do-Study-Act, який передав своєму учневі Уільяму Демінгу (нині цей підхід відомий як Цикл Демінга). Після завершення Другої світової війни компанія Toyota, відома своїми новаторськими методами, зокрема Lean і Kanban, запросила Демінга навчити своїх менеджерів.

З розвитком комп’ютерної галузі перед інженерами постала задача уніфікації методик розробки для підвищення рівня та систематизації цієї сфери. Однак, незважаючи на появу низки моделей управління, покращення якості роботи не відбулося. Замість цього виникли досить формалізовані методики, які негативно впливали на швидкість і якість продукту. Проєкти розробки займали багато часу, що призводило до застарівання формальних вимог ще до завершення розробки.

Тим часом невеликі команди, що уникали формалізованих методів, успішно випускали продукти швидше, якісніше та дешевше. Це стало революційним викликом бюрократичному підходу до розробки. Як зазначає Юрген Аппело у своїй книзі "Agile-менеджмент": "Еволюція виростила динозаврів, але вся їжа дісталася мурах".

Згодом лідери цього руху створили Agile Alliance і почали активно популяризувати методи гнучкого управління у світі, що призвело до виникнення цілої екосистеми конференцій, книг і практичних кейсів. За два десятиліття ця екосистема продовжує зростати.

Гнучка методологія управління була створена для вирішення численних проблем класичної водоспадної моделі (Waterfall), зокрема надмірного акценту на плануванні та затримок, що впливали на роботу інших команд. Щоб подолати ці проблеми, довелося повністю переглянути підхід до проєктної роботи, а не просто змінити окремі механіки.

"Гнучкість" полягає у здатності agile-команд адаптуватися до змінних умов, що досягається через три ключові аспекти:

1. Ітеративність. Замість тривалого планування та створення ідеальної версії продукту, agile-команда прагне якомога раніше випустити працездатний прототип і поступово його вдосконалювати. Хоча ітеративність може подовжити розробку, вона забезпечує швидке отримання робочого продукту.
2. Самоорганізація. У команді відсутні формальні керівники, що дозволяє уникнути тривалих погоджень і заощадити ресурси, зокрема час.
3. Взаємопроникнення знань. Кожен член agile-команди повинен мати базові знання суміжних спеціальностей, що підвищує крос-функціональність і підтримує високий рівень когнітивної активності.

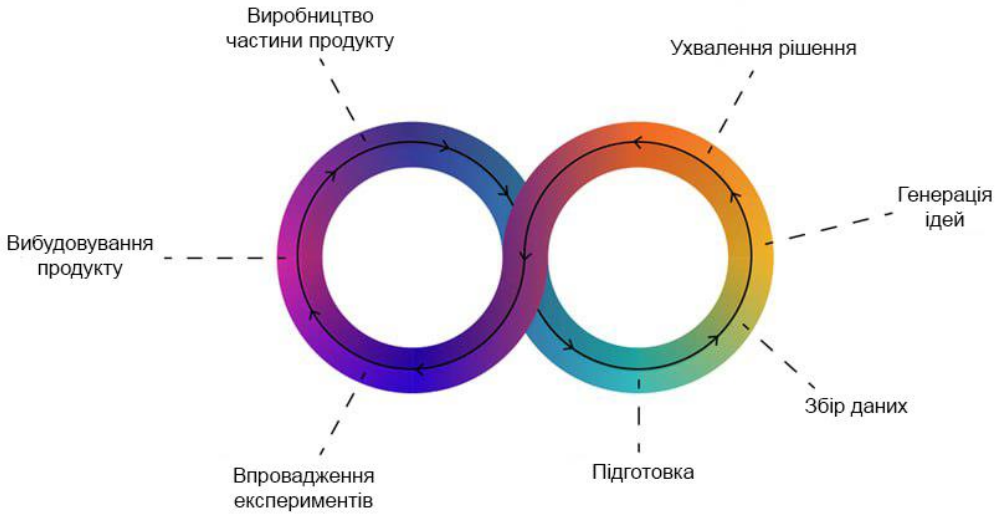

Рисунок 1.1 – циклічний процес розробки продукту

4 цінності та 12 принципів Agile-методології

Методологія — це сукупність методів і принципів, підкріплених теоретичними основами. Agile-методологія базується на цінностях, викладених у Agile Manifesto, а також на 12 принципах, що визначають її підходи.

Люди та взаємодія важливіші за процеси та інструменти. У центрі уваги Agile стоять люди, а не інструменти чи формальні процеси. Якщо в команді є елементи, що перешкоджають продуктивності — їх слід усунути. Члени команди самі обирають оптимальні методи організації, процеси й інструменти для своєї роботи, забезпечуючи таким чином максимальну ефективність.

Працюючий продукт важливіший за документацію. Це не означає, що в Agile не створюють документацію, але її обсяг і витрати на її підготовку значно зменшуються. Основна увага приділяється створенню функціонального продукту, а не деталізації кожного аспекту в документах.

Співпраця із замовником важливіша за узгодження умов контракту. Успішне виконання проєкту залежить від ефективної взаємодії із замовником, а не лише від дотримання умов контракту. Пріоритетом є збереження добрих стосунків із клієнтом, навіть якщо це вимагає гнучкості в умовах контракту.

Готовність до змін важливіша за проходження початкового плану. Навіть найдетальніший план потребує коригування під час виконання проєкту. Agile підхід передбачає постійну готовність адаптуватися до змін, що дозволяє командам залишатися ефективними в умовах динамічного середовища.

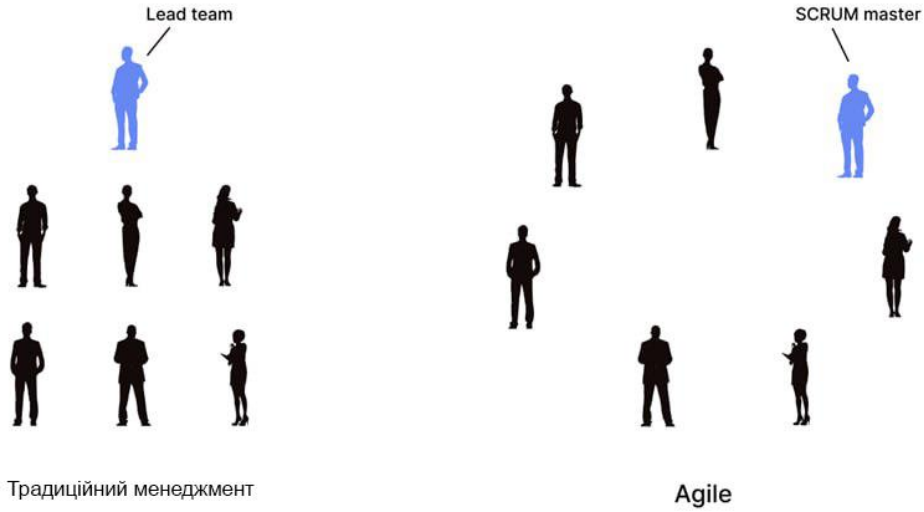

Рисунок 1.2 -

У Маніфесті Agile також викладено 12 принципів, які детальніше роз'яснюють ключові цінності. Розглянемо їх коротко:

1. Задоволення клієнта — найголовніше завдання розробки.
2. Готовність до змін у процесах забезпечує конкурентну перевагу продукту.
3. Програмне забезпечення, що працює, повинно регулярно доставлятися клієнту з періодичністю від 2 до 16 тижнів.
4. Співпраця між керівниками та розробниками має тривати протягом усього процесу розробки.
5. Основою проєктів є мотивовані люди, яким слід забезпечити належні умови роботи.
6. Найефективніший спосіб передачі інформації — це особисте

спілкування.

7. Успіх вимірюється працюючим програмним забезпеченням, а не витраченим часом чи ресурсами.
8. Гнучкість є основою сталого розвитку.
9. Особливу увагу слід приділяти технічній досконалості та якісному дизайну продукту.
10. Усунення зайвої роботи та спрощення процесів сприяє ефективності.
11. Самоорганізація команд забезпечує створення найкращих продуктів, уникаючи мікроменеджменту.
12. Регулярна оцінка та саморефлексія допомагають удосконалювати процеси.

Ці принципи є практичним втіленням цінностей Agile, які застосовуються для забезпечення ефективності команди та якості продукту.

Підходи, методи та інструменти Agile

У межах Agile-методології було розроблено та впроваджено низку інструментів, які сприяють досягненню її цілей.

Переваги Agile:

- Адаптивність до змін.
- Висока залученість команди.
- Дотримання термінів.
- Постійна взаємодія з клієнтом.
- Створення необхідного продукту.
- Зменшення ризиків завдяки ранньому виявленню проблем.

Недоліки Agile:

- Залежність від мотивації команди.

- Висока залежність від кваліфікації команди.
- Відсутність чіткого плану.
- Складність переходу від традиційних методів до гнучких і навпаки.

Застосування Agile

Хоча Agile було розроблено для управління проєктами в сфері розробки програмного забезпечення, він знайшов застосування і в інших галузях. Відомі проєкти, створені за Agile, включають:

- Spotify — платформа для стримінгу музики.
- Microsoft Azure — хмарна платформа.
- eBay — платформа для онлайн-аукціонів.

Agile у розробці

Agile-методологія народилася в сфері розробки програмного забезпечення і має свої особливості. Основний акцент робиться на короткі ітерації, адаптацію планів відповідно до нових вимог, використання зворотного зв'язку від клієнтів та регулярну самооцінку команди.

Agile дозволяє збирати метрики для покращення процесу розробки, зокрема:

- Velocity: кількість виконаних задач або story points за ітерацію.
- Cycle Time: час від створення задачі до її завершення.
- Lead Time: час від початку розробки до доставки функціоналу користувачам.
- Кількість багів, виявлених у кожній ітерації.

1.1.2 Scrum

Scrum — це методика організації спільного робочого процесу, яка базується на поетапній розробці та вдосконаленні продукту невеликою

командою спеціалістів різного профілю. Вона була розроблена програмістами Джеффом Сазерлендом і Кеном Швабером, які, вивчаючи методи роботи американських військових і спецпідрозділів, дійшли висновку, що успіх значною мірою залежить від якісної командної роботи. Сам термін "Scrum" походить із регбі, де він означає "сутичка" і символізує скоординовані дії команди для досягнення спільної мети. Спочатку методика використовувалася серед розробників програмного забезпечення, а згодом стала популярною у різних галузях бізнесу.

Scrum належить до сімейства гнучких методологій Agile, які передбачають адаптивний підхід до управління проєктами. Хоча ці поняття часто використовуються як синоніми, це не зовсім коректно. Scrum — це конкретна методика, що має свої правила, структури та етапи роботи, тоді як Agile є загальною філософією чи набором цінностей, які визначають підхід до розробки програмного забезпечення та інших проєктів.

Особливості Scrum

Scrum вирізняється командним підходом і специфічним розподілом обов’язків серед учасників. До процесу залучаються як співробітники компанії, так і бізнес-замовники, що сприяє більш ефективній комунікації та гнучкості в управлінні проєктом.

Мета методології Scrum

Ця методика відрізняється своєю гнучкістю і дозволяє командам експериментувати з різними підходами, що робить її ефективною для швидкої розробки нових продуктів. Вона особливо корисна в умовах невизначеності кінцевого результату або частих змін на ринку. Scrum допомагає командам поступово досягати поставлених цілей і контролювати ефективність роботи на кожному етапі проєкту.

Процес роботи Scrum-команди

Основною метою Scrum є забезпечення замовника бажаним продуктом у встановлені терміни з мінімальними витратами. Це досягається шляхом послідовного виконання кількох ключових етапів:

1. Розробка беклогу продукту: Власник продукту створює концепцію, враховуючи ринкові потреби і вимоги користувачів. На основі цього формується список завдань, упорядкованих за пріоритетами, що стає технічним завданням для команди.
2. Збір команди: Scrum-команда складається з невеликої групи спеціалістів (зазвичай 6-10 осіб) з різними навичками, які працюють на спільний результат. У команді відсутні ієрархічні структури, що забезпечує рівноправність і сприяє самоорганізації. Основні ролі включають:
    - Власник продукту: представляє інтереси замовника, забезпечує розробників необхідною інформацією і координує їх роботу відповідно до бізнес-вимог.
    - Scrum-майстер: стежить за дотриманням принципів Scrum, допомагає команді уникати перешкод і сприяє ефективній роботі.
    - Розробники: забезпечують технічну реалізацію проєкту, взаємодіючи один з одним для досягнення оптимальних результатів.
3. Планування спринтів: Спринт — це визначений період, протягом якого команда створює та вдосконалює окрему частину продукту. Кожен спринт починається з обговорення беклогу, на основі якого формується список завдань для виконання в межах цього циклу. Тривалість спринту зазвичай становить близько двох тижнів.

Після завершення кожного спринту команда оцінює свої досягнення і планує наступні кроки, враховуючи отриманий зворотний зв'язок. Це дозволяє гнучко реагувати на зміни і постійно вдосконалювати продукт.

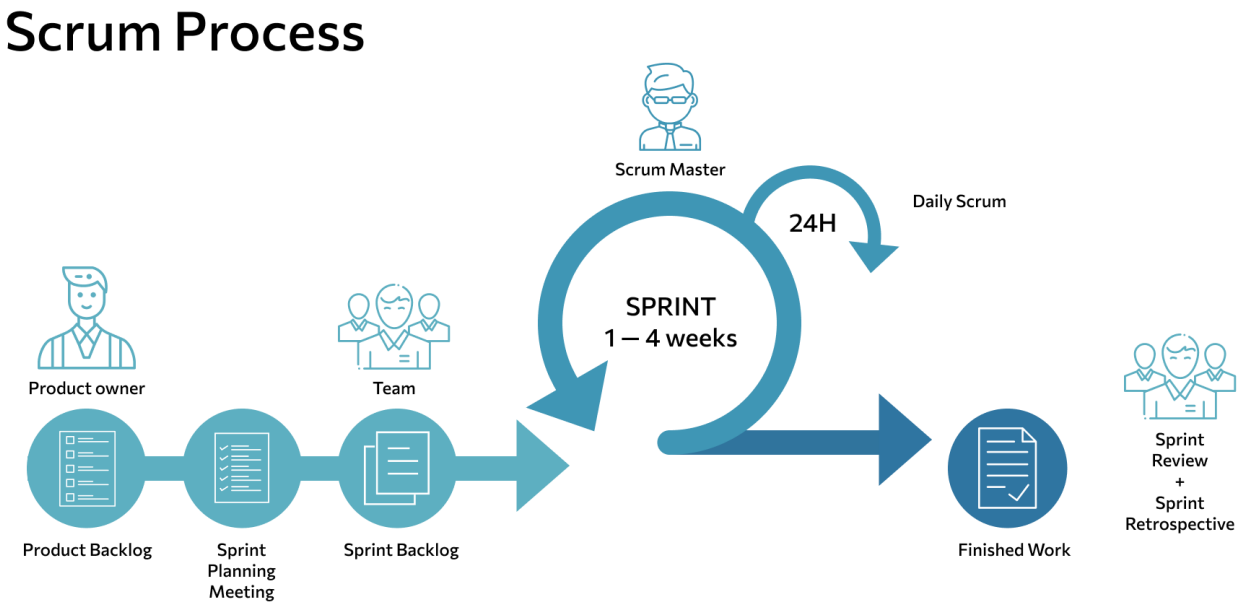

Рисунок 1.3 – Scrum Process

Scrum-мітинг, або стендап

Щодня команда збирається на коротку зустріч, що триває не більше 15 хвилин. Метою цієї зустрічі є обмін інформацією між членами команди щодо прогресу роботи. Кожен учасник відповідає на три ключові питання:

1. Що було зроблено з часу останньої зустрічі?
2. Що планується зробити сьогодні?
3. Що перешкоджає виконанню завдань?

На основі цих мікрозвітів Scrum-майстер аналізує, чи є проблеми в робочому процесі і як можна допомогти команді подолати перешкоди.

Scrum-дошка

Для візуалізації процесу команда використовує фізичні або програмні дошки, які поділяються на кілька колонок, що відображають стадії виконання завдань. Хоча кількість колонок може варіюватися, обов'язковими є три основні:

- Заплановані завдання.
- Завдання в активній роботі.
- Виконані завдання.

Scrum-дошка надає кожному члену команди можливість контролювати

прогрес власної роботи та загальний стан проекту.

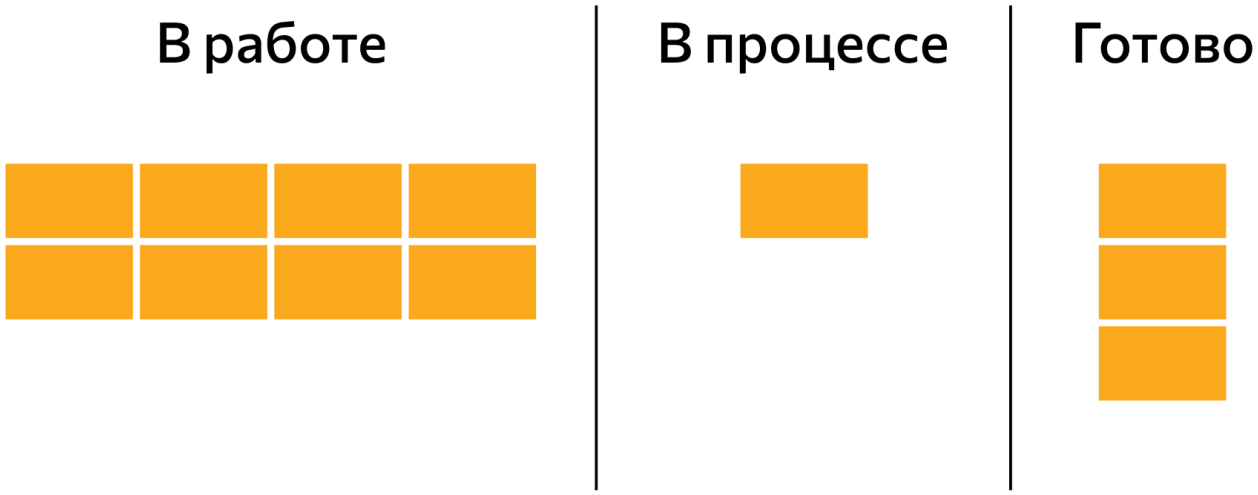

Рисунок 1.4 – дошка завдань Scrum

Підбиття підсумків спринту

Після кожного спринту проводиться оцінка процесу та тестування продукту. Якщо результати не відповідають очікуванням, команда коригує стратегію розробки або переглядає беклог. Збір даних для аналізу включає:

- **Velocity**: оцінка продуктивності команди на основі кількості виконаних завдань.
- **Burndown Chart**: діаграма, що відображає залишок роботи у спринті.
- **Відсоток виконаних завдань** від запланованих.
- **Кількість багів**, виявлених після завершення спринту.

Огляд результатів роботи над проєктом

Наприкінці спринту вся команда, включаючи власника продукту та Scrum-майстра, проводить демонстрацію результатів роботи. Кожен розробник презентує виконані завдання з беклогу. Після цього відбувається оцінка результатів, і вся команда демонструє загальний підсумок у вигляді працюючого продукту. Власник продукту ухвалює рішення щодо випуску продукту або необхідності подальших доопрацювань. Також проводиться ретроспектива робочого процесу, де команда обговорює свої спостереження, проблеми та взаємодію.

Принципи роботи Scrum-команди

Під час роботи за методикою Scrum команда дотримується таких принципів:

- **Постійне вдосконалення:** продукт і процеси постійно вдосконалюються завдяки самовдосконаленню команди.
- **Автономність:** кожен учасник відповідає за свою частину роботи та за загальний результат.
- **Кросфункціональність:** наявність у команді спеціалістів із різними навичками робить її самодостатньою.

Чим Scrum відрізняється від Kanban

Kanban також є частиною сімейства Agile, але має інший підхід до управління проєктами. Scrum — це структурований підхід із заданими етапами, тоді як Kanban фокусується на рівномірному розподілі роботи серед членів команди. У Scrum робота організована у спринтах, тоді як у Kanban завдання можуть з’являтися у будь-який момент.

Переваги Scrum:

- Команда працює короткими етапами, що підвищує швидкість роботи.
- Кожен член команди знає свої обов'язки, що підвищує відповідальність.
- Прозорий обмін інформацією робить процес більш відкритим і зрозумілим.
- Швидка реакція на зміни знижує ризики.

Недоліки Scrum:

- Не підходить для великих і складних проєктів через можливі проблеми з координацією.
- Вимагає високого рівня довіри у команді.

- Після тривалого періоду роботи може знижуватися продуктивність, що потребує переформатування команди.
- Потребує постійного зворотного зв'язку від замовника.

1.1.3 Waterfall (Каскадна модель)

Каскадна модель розробки (Waterfall) передбачає послідовне виконання етапів, при якому не допускається повернення на попередні стадії або пропускання етапів. У класичному варіанті, описаному В. Ройсом, модель включає сім основних етапів:

1. Формулювання вимог (Збір вимог):
   - Аналіз потреб замовника та користувачів.
   - Визначення цілей проєкту, специфікацій функціоналу та технічних вимог.
   - Підготовка документації, яка фіксує всі зібрані вимоги.

   Результат: Документ із вимогами до програмного забезпечення (SRS), що містить чіткий опис завдань і очікувань від продукту.
2. Проектування (Дизайн):
   - Архітектурне проектування: створення структури програмного забезпечення, вибір технологій, інструментів і підходів до розробки.
   - Детальне проектування: розробка схем модулів, бази даних і інтерфейсів.

   Результат: Технічна документація та архітектурні схеми системи.
3. Реалізація:
   - Програмування та написання коду відповідно до технічної документації.
   - Поділ задач між розробниками, створення модулів згідно зі специфікаціями.

   Результат: Робочий код, готовий до інтеграції та тестування.
4. Інтеграція (Здійснення):

- Об'єднання модулів і компонентів системи в єдине ціле.
- Налагодження зв'язків між підсистемами і перевірка їх взаємодії.

Результат: Цілісна програмна система, готова до тестування.

5. Тестування:
   - Перевірка відповідності продукту вимогам.
   - Виявлення та виправлення дефектів.
   - Виконання різних видів тестів (функціонального, навантажувального, інтеграційного тощо).

Результат: Тестові звіти та список знайдених і виправлених помилок.

6. Встановлення (Реліз):
   - Передача готового продукту замовнику або користувачам.
   - Встановлення системи на цільових пристроях і, за необхідності, навчання користувачів.

Результат: Готовий продукт, доступний для використання.

7. Підтримка:
   - Виправлення дефектів, які виявляються під час експлуатації.
   - Додавання нових функцій та адаптація до змін у вимогах користувачів.
   - Підтримання стабільної роботи продукту протягом його життєвого циклу.

Результат: Постійно вдосконалений і стабільний продукт.

Особливості водоспадної моделі розробки

Головна відмінність Waterfall від інших методологій полягає у відсутності гнучкості. У той час як методології Agile або Scrum дозволяють паралельне виконання етапів та зміни у процесі, Waterfall дотримується суворої послідовності. Це означає, що кожен етап повинен бути завершений до початку наступного, що унеможливлює проміжні перевірки.

Значну увагу у Waterfall приділяють етапу проектування, оскільки помилки на цьому етапі можуть призвести до провалу всього проєкту. Недоліки

виявляються лише на етапі тестування, коли продукт вже повністю готовий. Це можна порівняти зі створенням літака: якщо проектувальники неправильно розрахували міцність конструкції, це стане відомо лише під час випробувань.

Як працює Waterfall

Розберемо докладніше, як улаштовані етапи роботи в каскадній моделі розробки, на прикладі комп'ютерної гри.

1. Визначення вимог. Технічних (по суті — складання ТЗ для програмістів), фінансових (визначення бюджету) тимчасових (терміни реалізації проекту) і т.д. 24 місяці. Тільки докладніше.

2. Проектування. На цьому етапі всі вимоги упаковуються у документацію. Для програмістів це буде логіка гри, дизайн героїв та багато іншого. Проектування — один із найтриваліших етапів розробки за каскадною моделлю. Про ціну помилки ми вже говорили.

3. Реалізація. Власне, написання коду документації, розробленої на попередніх етапах.

4. Здійснення - з'єднання елементів програми в єдине ціле.

5. Тестування: перевірка готової гри щодо помилок. Правильніше було б назвати цей етап пусконалагоджувальним: всі виявлені баги усуваються або тестувальниками, або програмістами відділу тестування. Ніхто нічого не відправляє на доопрацювання: пам'ятаємо, що повернення на попередні етапи не передбачено.

6. Встановлення. У широкому сенсі — випуск товару ринку та його інсталяція на пристрої кінцевих користувачів (якщо мова про коробочної версії) чи доступом до хмарі (якщо мова про надання ПО по SaaS-моделі).

7. Технічна підтримка. Супровід продукту в експлуатації: випуск оновлень, усунення помилок тощо.

Переваги та недоліки водоспадної моделі

Усі біди та недоліки каскадної методології випливають із того, що етапи розробки йдуть послідовно. Почну з того, за що підхід критикують та застосовують обмежено.

• Найголовніший мінус – неможливість нічого змінити, коли процес розробки запущено. Якщо замовник на півдорозі схаменувся і вирішив додати до ПЗ новий функціонал, доведеться все відновити. Гнучкість нуль.

• Це довго та дорого, якщо порівнювати з гнучкими методологіями.

• Один продукт – один проект. Розробки не можна поширити інше ПЗ. Жодної універсальності або можливість використовувати проект як шаблон. Звідси…

• …довго та дорого. Простий приклад: за одним проектом будинку можна збудувати будь-яку кількість будинків. А тут так не вийде: одна програма — один проект, тож дорожче та довше.

• Потрібно багато персоналу: розробники, проектувальники, аналітики, програмісти, тестувальники тощо.

• Критична залежність успіху всього проекту від якості проектування та розробки документації.

• Тестування проводиться наприкінці. Якщо виявлено фатальні помилки, переробляється все від початку.

• Надмірна бюрократизація процесу розробки. Часто на перший план, замість якості ПЗ та задоволеності замовника, виходять такі параметри, як трудовитрати, трудомісткість та інші речі, що не мають відношення до якості продукту.

• Немає місця для імпровізації. Якщо у когось із виконавців виникла ідея щодо покращення продукту, реалізувати її не вдасться.

Проте підхід має сильні сторони:

• Команда та окремі фахівці завжди знають, що робити. Вся робота детально та покроково розписана.

• Завжди відомі терміни та бюджет проекту.

• Прозорість роботи та повне розуміння з боку замовника.

• Взаємозамінність фахівців. Завдяки докладній документації можна реалізувати проект силами будь-якої компетентної команди.

Коли варто використовувати модель Waterfall

Незважаючи на критику та недоліки, є випадки, коли застосування Waterfall цілком виправдане. Наприклад:

- Складні та нестандартні проекти, до яких неможливий шаблонний підхід та всю технічну документацію потрібно розробляти з нуля.
- Умови проекту свідомо незмінні.
- Під час виконання проекту можлива зміна підрядника, виконання етапів різними підрядниками, участь різних спеціалістів. По осудній документації розпочати проект можуть одні фахівці, а закінчити інші.
- До результату проекту висуваються чіткі вимоги
- Будь-які інші випадки, коли гнучкі підходи на кшталт Scrum або Agile не працюють або не влаштовують замовника.

Приклади використання каскадної методології

Слід зазначити, що каскадна модель (Waterfall) не завжди застосовується у своїй класичній формі. Найчастіше під час розробки використовується комбінація різних методологій, які включають елементи Waterfall. Наприклад, ітерації Agile можуть виступати як "мікроводоспади", що мають власну документацію, жорстку структуру та інші особливості.

Наприкінці минулого століття Waterfall широко використовувалася такими гігантами, як IBM, Microsoft, Toyota, та іншими великими компаніями. Ця методологія залишалася домінуючою через її структурованість і передбачуваність.

Навіть сьогодні каскадна модель широко застосовується в таких галузях, як медицина, банківський сектор та державні замовлення. Це пояснюється консервативністю цих сфер та необхідністю чіткого визначення термінів і бюджету проєктів. Agile та інші гнучкі методології часто не відповідають формальним вимогам замовників, які потребують детальної документації та суворого дотримання плану.

1.1.4 Kanban

Канбан — це метод для розробки продуктів, який допомагає налагодити

поточні процеси і перевантажити команду. Незавершені завдання не простоюють і потоком рухаються ланцюжком створення продукту або його підтримки.

В основі kanban — Agile та гнучка розробка.

Для ефективного впровадження методології Kanban у команді розробників програмного забезпечення важливо відповідним чином систематизувати їхню роботу. Це дозволить виконувати завдання в рівномірному темпі та оптимізувати керування робочим процесом. Ось як можна структурувати процес Kanban.

Візуалізуйте. Для початку наочно уявіть робочий процес команди на дошці Kanban, фізичній чи віртуальній. Відобразіть кожен етап розробки — від створення завдань до завершення.

Стандартизуйте. Визначте та стандартизуйте етапи робочого процесу відповідно до методів та вимог вашої команди. Зазвичай використовують етапи «До виконання», «У роботі» та «Готово», але за необхідності їх можна адаптувати під особливості вашого робочого процесу.

Визначте блокери та залежності. Важливо, щоб дошкою Kanban можна було миттєво виявити блокери і залежності. Така прозорість дозволяє без зволікання вирішувати проблеми та запобігати перебоям у робочому процесі.

Обмежте обсяги незавершеної роботи. Передбачте обмеження кожного етапу, щоб уникати перевантажень і підтримувати стабільний робочий процес. Це допоможе оптимізувати розподіл ресурсів та рідше працювати над кількома завданнями одночасно, що сприяє підвищенню продуктивності.

Заохочуйте спільну роботу. Розвивайте культуру співробітництва у команді, щоб учасники спільно усували вузькі місця та просувалися по етапах робочого процесу в рівномірному темпі. Це сприяє ефективному та швидкому виконанню завдань.

Використовуйте картки Kanban. Подайте кожне завдання на дошці у

вигляді картки Kanban з важливими відомостями, такими як опис завдання, виконавець та передбачуваний час виконання. Картки Kanban допомагають візуально відстежувати прогрес та сприяють прозорості у команді.

Структурувавши процес Kanban таким чином, ви зможете оптимізувати розробку програмного забезпечення, зміцнити співпрацю в команді та досягти максимальної ефективності в управлінні завданнями.

Чотири базові принципи Kanban

1. **Відштовхуйтеся від того, що у вас є зараз**

Методологія не закликає миттєво змінювати структуру компанії та ролі працівників. Навпаки, потрібно впроваджувати зміни до вже наявної системи.

2. **Прагнути поетапних, постійних та еволюційних змін**

Інакше кажучи — йти до великої мети маленькими кроками. На перший погляд може здатися, що глобальні зміни принесуть більше користі та прибутку. Але треба пам'ятати, що вони також принесуть величезні ризики.

Поступовий рух до мети — більш гнучкий та безпечний підхід.

3. **Поважайте потокові процеси та ролі**

Потрібно зберегти те, що працює добре. Це стосується і стосунків, і посад, і процесів. Зв'язки з людьми допоможуть отримати підтримку змін, а налагоджені процеси удосконалити нестабільні.

4. **Підтримувати лідерство на всіх рівнях**

Прагнути бути лідерами та пропонувати зміни мають працівники на всіх рівнях, а не лише менеджмент.

Етапи:

Kanban — це насамперед візуалізація. Для візуалізації використовують дошку та набір різнокольорових карток. Один колір — один виконавець чи процес. Усі учасники команди будь-коли можуть перевірити стан будь-якого

завдання.

Такі дошки можна використовувати і для особистого тайм-менеджменту, і проєктного планування. Найпростіша канбан-дошка — це три колонки: “To Do” (Зробити), “In Progress” (У роботі), “Done” (Готово).

Рисунок 1.5 – дошка завдань Kanban

Кожен проєкт поділено різні етапи. Скільки етапів — стільки й стовпців у канбан-дошці. Ось як виглядає kanban для IT-проєкту:

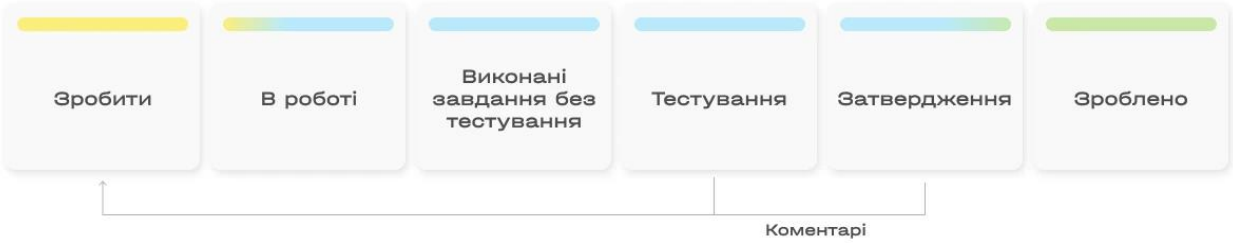

Рисунок 1.6 – проект поділений на етапи Kanban

Дані для збору

На kanban-дошці відображено всі процеси проєкту. Так їх просто проаналізувати і вчасно помітити проблемні місця. Метрики які використовуються для цього є наступні:

- **Cycle Time**: час виконання кожної задачі.
- Відсоток задач, що переходять з одного етапу на інший.
- **Lead Time**: час з моменту постановки задачі до її завершення.

- **Task Completion Rate**: кількість завершених задач за певний період часу.

Відомі проекти, розроблені за Kanban:

- Toyota — виробничі процеси, де Kanban було вперше впроваджено.
- Zara — логістика та управління постачаннями.
- Netflix — управління потоками розробки контенту.

Переваги:

- Візуалізація робочого процесу:

Картки на дошці допомагають побачити прогрес та проблеми у русі завдань.

- Швидка адаптація команди:

Принцип Kanban – поступове покращення існуючих процесів. Відразу не потрібно нічого міняти, можна розпочати роботу з тим, що є. Тому команда не матиме простою під час перебудови.

- Багато метрик:

За допомогою метрик можна відстежити, на якому етапі гальмує робочий процес та в чому причини. Часто можна дійти несподіваних висновків, наприклад, знайти проблеми там, де робота здавалася добре налаштованою.

- Рівномірний розподіл завантаження команди:

Відразу видно, коли в якійсь колонці стало дуже багато карток. Це означає, що виконавцям потрібна допомога чи додатковий час, щоб розібратися із завданнями.

Недоліки:

- Складнощі з плануванням:

На початковому етапі впровадження Kanban важко прогнозувати

результати. Без аналізу метрик не вдасться розрахувати скільки часу потрібно команді на завершення завдання.

- Складнощі у спілкуванні із замовниками:

Представникам бізнесу найчастіше потрібні точні терміни завершення завдань. Тому замовники можуть не зрозуміти прогнозів із ймовірністю, яку розраховують на основі метрик та класів обслуговування.

- Підходить не для всіх:

Гнучке планування Kanban може не підійти компаніям з квартальними звітами і системою жорстких KPIs.

1.2 Метрики для оцінки роботи команд розробки

Метрики допомагають оцінити продуктивність, якість роботи та ефективність команд розробки програмного забезпечення. Використання цих інструментів сприяє виявленню вузьких місць у процесі розробки та оптимізації загального робочого процесу. Нижче наведені основні метрики та їх значення.

Velocity (Швидкість виконання)

- **Опис**: Кількість завершених задач або story points за одну ітерацію чи спринт.
- **Важливість**: Дає змогу командам оцінити свою продуктивність і планувати майбутні спринти на основі минулих результатів.
- **Дані**:
    - Кількість виконаних задач.
    - Тривалість спринтів.

Lead Time (Час надання)

- **Опис**: Час від моменту постановки задачі до її завершення та доставки користувачеві.

- **Важливість**: Показує, наскільки швидко команда може реагувати на зміни у вимогах та доставляти функціонал.
- **Дані**:
    - Дата створення задачі.
    - Дата її завершення.

Cycle Time (Час циклу)

- **Опис**: Час, необхідний для завершення задачі з моменту початку роботи над нею до її завершення.
- **Важливість**: Дає змогу виявити вузькі місця в процесі розробки та оцінити ефективність робочого потоку.
- **Дані**:
    - Дата початку роботи над задачею.
    - Дата завершення задачі.

Defect Density (Щільність дефектів)

- **Опис**: Кількість виявлених дефектів або багів на одиницю коду (наприклад, на 1000 рядків коду).
- **Важливість**: Дозволяє оцінити якість програмного забезпечення та ефективність тестування.
- **Дані**:
    - Кількість дефектів.
    - Загальна кількість рядків коду (LOC — Lines of Code).

Customer Satisfaction (Задоволеність клієнтів)

- **Опис**: Рівень задоволення користувачів продуктом, виміряний за допомогою анкетування, опитувань або збору зворотного зв’язку.
- **Важливість**: Допомагає зрозуміти, наскільки продукт відповідає

очікуванням кінцевих користувачів.

- **Дані**:
    - Відгуки клієнтів.
    - Оцінки NPS (Net Promoter Score).

Burndown Chart

- **Опис**: Графік, що відображає залишок роботи до завершення спринту або проекту.
- **Важливість**: Дозволяє відслідковувати прогрес команди та вчасно коригувати плани.
- **Дані**:
    - Кількість запланованих задач.
    - Кількість виконаних задач.

Task Completion Rate (Рівень виконання задач)

- **Опис**: Частка задач, виконаних за визначений період часу.
- **Важливість**: Допомагає оцінити загальну продуктивність команди та виявити затримки в процесі.
- **Дані**:
    - Кількість завершених задач.
    - Загальна кількість задач, запланованих на період.

Code Coverage (Покриття коду)

- **Опис**: Відсоток коду, перевіреного за допомогою тестів (unit tests, integration tests тощо).
- **Важливість**: Показує якість тестування та рівень покриття функціоналу.
- **Дані**:
    - Загальна кількість рядків коду.

- Кількість рядків коду, перевірених тестами.

Висновок

Метрики дозволяють отримувати кількісні показники ефективності роботи команд розробників. Вони сприяють своєчасному виявленню проблем, покращенню робочих процесів і підвищенню якості кінцевого продукту. Використання метрик має бути адаптованим до потреб проекту, щоб забезпечити найбільш релевантні результати.

Висновок до розділу 1

У першому розділі магістерської роботи було детально розглянуто теоретичні основи управління розробкою програмного забезпечення. Основна увага приділялася аналізу сучасних методологій, які використовуються в індустрії, а також їхніх сильних і слабких сторін. Під час дослідження було виявлено, що кожна з розглянутих методологій має свої унікальні особливості та сферу застосування. Наприклад, Scrum є гнучкою методологією, яка дозволяє командам швидко адаптуватися до змін і забезпечує високу ефективність у динамічних проектах. Водночас, Kanban вирізняється своєю простотою та фокусом на постійному вдосконаленні процесів, що робить його ідеальним для підтримки існуючих систем. З іншого боку, Waterfall демонструє структурованість і передбачуваність, що є критично важливим для проєктів із чітко визначеними вимогами. Однак, незважаючи на переваги кожного підходу, їхнє застосування значною мірою залежить від конкретного контексту проєкту, потреб команди та очікувань клієнтів. Підсумовуючи, перший розділ формує теоретичну базу, яка стане основою для подальшого аналізу та практичного застосування методологій у наступних частинах роботи.

Розділ 2 Моделювання

У цьому розділі розглядаються імітаційні моделі, їх метрики та алгоритми за якими вони працюють у цих моделях. Також опис програмного продукта, який буде проганятись у цих моделях.

2.1 Імітаційна модель Scrum

Scrum — це ітеративна методологія, яка використовує спринти для організації роботи. У моделі для проекту FinancePlus Scrum реалізується через симуляцію виконання завдань за фіксованими ітераціями з урахуванням їхньої складності, часу виконання, тестового покриття, а також щільності дефектів. Модель інтегрує метрики для оцінки ефективності роботи команди, такі як Velocity та Defect Density.

2.1.1 Основні елементи Scrum

1. **Завдання (Task):**

Назва, оцінка тривалості, покриття тестами, кількість дефектів, час початку і завершення.

2. **Спринти:**

Фіксована кількість завдань у кожному спринті.

3. **Метрики:**

Velocity: кількість завдань, завершених у спринті.

Defect Density: кількість дефектів на завдання.

4. **Ресурси:**

Середовище симуляції (simpy) використовується для обробки завдань у реальному часі.

5. **Excel-звіти:**

Дані про виконані завдання та метрики зберігаються в Excel.

2.1.2 Опис логіки моделі

Модель Scrum базується на циклічному виконанні завдань, що називаються спринтами. Кожен спринт має фіксовану тривалість і кількість завдань, які команда повинна завершити. Після кожного спринту підводяться підсумки, і оцінюється продуктивність команди, що дає змогу постійно вдосконалювати процес.

1. **Управління беклогом**

- Завдання додаються до беклогу, де вони очікують свого виконання.
- На початку кожного спринту команда обирає певну кількість завдань для виконання.
- Це дозволяє керувати пріоритетами та гнучко реагувати на зміни в проекті.

2. **Симуляція випадкових відхилень**

- Для кожного завдання моделюються випадкові відхилення тривалості. Це враховує реальні ситуації, коли виконання може зайняти більше або менше часу через непередбачені обставини.
- Логіка відхилень базується на генерації випадкового числа у межах ±20% від базової тривалості завдання.

3. **Оцінка дефектів**

- Після завершення завдання моделюється кількість дефектів, що виникають. Це залежить від рівня покриття тестами: чим нижче покриття, тим більша ймовірність дефектів.
- Дефектність завдання обчислюється як відношення кількості дефектів до фактичної тривалості завдання.

4. **Зворотний зв'язок**

- Модель забезпечує постійний зворотний зв'язок через метрики, такі як швидкість виконання (velocity), густина дефектів (defect density) і

задоволеність клієнтів (customer satisfaction).

- Ці метрики дозволяють команді аналізувати свої досягнення та знаходити шляхи для вдосконалення.

5. **Ітеративне вдосконалення**

- Кожен спринт надає можливість переглянути виконану роботу та внести необхідні корективи для наступних ітерацій.
- Це створює цикл безперервного вдосконалення, що є ключовим принципом Scrum.

6. **Оцінка задоволеності клієнтів**

- Задоволеність клієнтів обчислюється на основі кількості дефектів у кожному завданні.
- Висока густина дефектів знижує задоволеність, тоді як менша кількість дефектів підвищує її.

2.1.3 Метрики Scrum

1. **Velocity (швидкість виконання):**

Показує кількість завдань, завершених за один спринт. Для підвищення ефективності спринтів доцільно використовувати формули прогнозування навантаження на команду. Наприклад:

$$Velocity = \frac{\text{Кількість завершених задач(у } story\ point)}{\text{Кількість спринтів}}$$

Ця формула дозволяє оцінити середню продуктивність команди та планувати обсяг задач для майбутніх спринтів. Приклад:

Якщо команда виконала 50 story points за 5 спринтів, то її velocity становить:

$$Velocity = \frac{50}{5} = 10\ story\ point \text{ на спринт}$$

Таким чином, у наступний спринт можна запланувати приблизно 10 story points.

Velocity
3
3
3
3
3

Рисунок – приклад отриманого Velocity

2. **Defect Density (щільність дефектів):**

Кількість дефектів, розподілена на завдання. Визначається формулою:

$$Defect\ Density = \frac{\text{Загальна кількість дефектів у спринті}}{\text{Кількість завершених завдань}}$$

Defect Density
0.333333333
0
0
0.333333333
0.333333333

Рисунок – приклад отриманого Defect Density

3. **Задоволеність клієнтів:**

Відображає рівень задоволеності на основі кількості дефектів

$$Customer\ Satisfaction = \max(0, 100 - (Defect\ Density * 100))$$

4. **Загальний час виконання:**

Сумарний час, витрачений на виконання всіх завдань проекту.

Рисунок – приклад отриманого загального часу

5. **Покриття тестами:**

Відсоток покриття тестами для кожного завдання, що впливає на ймовірність появи дефектів.

2.3.4 Приклад роботи Scrum у проекті FinancePlus

1. **Вхідні дані:**

Кількість фаз: У проекті було додано 26 фаз (завдань), кожна з яких мала заплановану тривалість.

Деталі завдань: Завдання включали різні аспекти розробки та тестування, наприклад, "Розробка архітектури проєкту," "Дизайн основного інтерфейсу," "Юніт-тестування основних функцій" тощо..

2. **Реалізація:**

Послідовне виконання: Завдання виконувалися послідовно по спринтах. У кожному спринті виконувалося до п'яти завдань.

Тривалість фаз: Для кожного завдання була визначена базова тривалість, але фактична тривалість варіювалася в межах ±20% від запланованої. Це моделювалося методом simulate_task_duration, який додає або віднімає випадкову величину, що дорівнює 20% базової тривалості.

Дефекти: Дефекти моделювалися на основі тестового покриття. Завдання з нижчим покриттям мали вищу ймовірність дефектів.

3. **Результати:**

Загальний час: Загальний час виконання проекту обчислювався як сума фактичної тривалості всіх завдань. Це значення було збережено у змінній total_time_spent.

Defect Density: Показник щільності дефектів (defect_density) обчислювався для кожного завдання і записувався у відповідний список. Показник залежав від кількості дефектів і тривалості завдання.

Customer Satisfaction: Рівень задоволеності клієнтів (customer_satisfaction) обчислювався на основі щільності дефектів і варіювався від 0 до 100.

Файл з часом виконання: results_time_Scrum.xlsx містить назви завдань, їхню заплановану та фактичну тривалість.

Файл з метриками: results_metrics_Scrum.xlsx містить назви завдань, їхню щільність дефектів і рівень задоволеності клієнтів.

```
=== Початок спринту 1 ===
[0] Виконується завдання: Розробка архітектури проєкту (Оцінка: 20 годин, Фактичний час: 23 годин)
[23] Завдання 'Розробка архітектури проєкту' завершено. Дефекти: 0, Густина дефектів: 0.00
[23] Виконується завдання: Дизайн основного інтерфейсу (Оцінка: 32 годин, Фактичний час: 37 годин)
[60] Завдання 'Дизайн основного інтерфейсу' завершено. Дефекти: 0, Густина дефектів: 0.00
[60] Виконується завдання: Розробка іконок та візуальних елементів (Оцінка: 24 годин, Фактичний час: 22 годин)
[82] Завдання 'Розробка іконок та візуальних елементів' завершено. Дефекти: 0, Густина дефектів: 0.00
[82] Виконується завдання: Дизайн графіків і діаграм (Оцінка: 24 годин, Фактичний час: 20 годин)
[102] Завдання 'Дизайн графіків і діаграм' завершено. Дефекти: 2, Густина дефектів: 0.08
[102] Виконується завдання: Дизайн сповіщень та повідомлень (Оцінка: 20 годин, Фактичний час: 23 годин)
[125] Завдання 'Дизайн сповіщень та повідомлень' завершено. Дефекти: 2, Густина дефектів: 0.10
=== Завершення спринту 1 ===

=== Початок спринту 2 ===
[125] Виконується завдання: Розробка головного меню та налаштувань (Оцінка: 40 годин, Фактичний час: 37 годин)
[162] Завдання 'Розробка головного меню та налаштувань' завершено. Дефекти: 3, Густина дефектів: 0.07
[162] Виконується завдання: Розробка екрану додавання транзакцій (Оцінка: 40 годин, Фактичний час: 48 годин)
[210] Завдання 'Розробка екрану додавання транзакцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[210] Виконується завдання: Створення функції додавання транзакцій (Оцінка: 40 годин, Фактичний час: 36 годин)
[246] Завдання 'Створення функції додавання транзакцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[246] Виконується завдання: Категоризація транзакцій (Оцінка: 24 годин, Фактичний час: 22 годин)
[268] Завдання 'Категоризація транзакцій' завершено. Дефекти: 2, Густина дефектів: 0.08
[268] Виконується завдання: Збереження даних користувача (Оцінка: 20 годин, Фактичний час: 18 годин)
[286] Завдання 'Збереження даних користувача' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 2 ===

=== Початок спринту 3 ===
[286] Виконується завдання: Побудова графіків витрат (Оцінка: 40 годин, Фактичний час: 41 годин)
[327] Завдання 'Побудова графіків витрат' завершено. Дефекти: 0, Густина дефектів: 0.00
[327] Виконується завдання: Автоматичне формування звітів (Оцінка: 24 годин, Фактичний час: 21 годин)
[348] Завдання 'Автоматичне формування звітів' завершено. Дефекти: 0, Густина дефектів: 0.00
[348] Виконується завдання: Система нагадувань про регулярні платежі (Оцінка: 24 годин, Фактичний час: 20 годин)
[368] Завдання 'Система нагадувань про регулярні платежі' завершено. Дефекти: 3, Густина дефектів: 0.12
[368] Виконується завдання: Інтеграція з банківськими API (Оцінка: 60 годин, Фактичний час: 70 годин)
[438] Завдання 'Інтеграція з банківськими API' завершено. Дефекти: 0, Густина дефектів: 0.00
[438] Виконується завдання: Реалізація цілей збережень (Оцінка: 32 годин, Фактичний час: 27 годин)
[465] Завдання 'Реалізація цілей збережень' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 3 ===
```

```
=== Початок спринту 4 ===
[465] Виконується завдання: Система захисту та шифрування даних (Оцінка: 20 годин, Фактичний час: 21 годин)
[486] Завдання 'Система захисту та шифрування даних' завершено. Дефекти: 0, Густина дефектів: 0.00
[486] Виконується завдання: Налаштування багатофакторної автентифікації (Оцінка: 24 годин, Фактичний час: 29 годин)
[515] Завдання 'Налаштування багатофакторної автентифікації' завершено. Дефекти: 1, Густина дефектів: 0.04
[515] Виконується завдання: Юніт-тестування основних функцій (Оцінка: 40 годин, Фактичний час: 36 годин)
[551] Завдання 'Юніт-тестування основних функцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[551] Виконується завдання: Інтеграційне тестування (Оцінка: 40 годин, Фактичний час: 47 годин)
[598] Завдання 'Інтеграційне тестування' завершено. Дефекти: 0, Густина дефектів: 0.00
[598] Виконується завдання: Стрес-тестування системи (Оцінка: 40 годин, Фактичний час: 37 годин)
[635] Завдання 'Стрес-тестування системи' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 4 ===

=== Початок спринту 5 ===
[635] Виконується завдання: Налагодження та оптимізація продуктивності (Оцінка: 40 годин, Фактичний час: 39 годин)
[674] Завдання 'Налагодження та оптимізація продуктивності' завершено. Дефекти: 0, Густина дефектів: 0.00
[674] Виконується завдання: Інтеграція Google API для синхронізації даних (Оцінка: 16 годин, Фактичний час: 16 годин)
[690] Завдання 'Інтеграція Google API для синхронізації даних' завершено. Дефекти: 2, Густина дефектів: 0.12
[690] Виконується завдання: Інтеграція платіжних систем (Оцінка: 16 годин, Фактичний час: 14 годин)
[704] Завдання 'Інтеграція платіжних систем' завершено. Дефекти: 0, Густина дефектів: 0.00
[704] Виконується завдання: Налаштування аналітичних інструментів (Оцінка: 32 годин, Фактичний час: 34 годин)
[738] Завдання 'Налаштування аналітичних інструментів' завершено. Дефекти: 0, Густина дефектів: 0.00
[738] Виконується завдання: Впровадження системи PUSH-повідомлень (Оцінка: 24 годин, Фактичний час: 21 годин)
[759] Завдання 'Впровадження системи PUSH-повідомлень' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 5 ===

=== Початок спринту 6 ===
[759] Виконується завдання: Додавання рейтингової системи користувачів (Оцінка: 32 годин, Фактичний час: 38 годин)
[797] Завдання 'Додавання рейтингової системи користувачів' завершено. Дефекти: 0, Густина дефектів: 0.00
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
=== Завершення спринту 6 ===
```

```
Результати часу виконання збережено у файл results_time_Scrum.xlsx

Результати метрик збережено у файл results_metrics_Scrum.xlsx

Кількість виконаних завдань: 26

Загальний час, витрачений на проект Scrum: 797 годин
```

Рисунок 2.1 – результат запуску моделі Scrum

4. Підсумок:

Моделювання 26 фаз: У моделі були змодельовані 26 фаз із різними рівнями тестового покриття, що дозволило отримати різноманітні значення метрик.

Оцінка ризиків: Отримані метрики дали змогу оцінити ключові ризики проекту, такі як ймовірність появи дефектів та вплив їх на задоволеність клієнтів.

2.2 Імітаційна модель Kanban

Модель орієнтована на управління потоком завдань без жорсткого часового обмеження. На відміну від Scrum, у Kanban завдання виконується у безперервному процесі, що дозволяє швидко адаптуватись до змін і вимогах. Основна мета моделі – мінімізація часу виконання завдань і оптимізація завантаження команди через обмеження кількості одночасно виконуваних завдань(WIP – Work in Progress).

2.2.1 Основні елементи Kanban

1. **Беклог:**

Містить список усіх завдань, які ще не розпочаті

2. **Робота в процесі:**

Набір завдань, які на даний момент перебувають у роботі. Розмір WIP обмежений, щоб уникнути перевантаження команди.

3. **Виконання завдання:**

Список завдань, які можуть успішно завершені й виведені з процесу

4. **Ресурси:**

Для виконання завдань використовується обмежена кількість ресурсів, що моделюється за допомогою бібліотеки simpy.

2.2.2 Опис логіки моделі

1. **Додавання завдань у беклог:**

Модель Scrum базується на циклічному виконанні завдань, що називаються спринтами. Кожен спринт має фіксовану тривалість і кількість завдань, які команда повинна завершити. Після кожного спринту підводяться підсумки, і оцінюється продуктивність команди, що дає змогу постійно вдосконалювати процес.

2. **Симуляція випадкових відхилень:**

Для кожного завдання симулюються випадкові відхилення тривалості

виконання в межах ±20% від базової тривалості, щоб врахувати реалістичні зміни у процесі роботи.

3. **Обмеження завдань у роботі:**

В роботі одночасно можуть бути три завдання, що відповідає обмеженню WIP (Work In Progress) в Kanban. Це дозволяє зменшити навантаження на команду і уникнути перевантаження системи.

4. **Виконання завдань:**

Завдання переміщуються з беклогу до списку завдань у роботі, виконуються, а після завершення переходять до списку завершених завдань. Після завершення кожного завдання генеруються метрики: густина дефектів і задоволеність клієнтів.

5. **Запис результатів:**

Час виконання і метрики кожного завдання записуються у відповідні файли Excel для подальшого аналізу.

6. **Оптимізація потоку:**

Канбан дозволяє постійно оптимізувати потік завдань, скорочуючи час виконання і підвищуючи якість продукту через регулярний зворотний зв'язок і оновлення пріоритетів завдань у беклозі.

2.2.3 Метрики Kanban

Для оцінки продуктивності моделі використовуються такі метрики:

1. **Cycle Time (час циклу):**

Визначає, скільки часу потрібно для виконання завдання від його початку до завершення. За його допомогою можна оптимізувати процес за допомогою розрахунку часу циклу (Cycle Time):

$$Cycle\ Time = \frac{\text{Час виконання задачі}}{\text{Кількість задач у роботі}}$$

Якщо команда працює над 4 задачами й витрачає на них 40 годин, то середній час циклу становить:

$$Cycle\ Time = \frac{40}{4} = 10 \text{ годин на задачу}$$

Цей показник допомагає оцінити, скільки часу потрібно для завершення кожної задачі та оптимізувати кількість задач у роботі (Work in Progress).

2. **Lead Time (час виконання):**

Враховує загальний час від додавання завдання у беклог до його завершення.

3. **Throughput (пропускна здатність):**

Визначає кількість завдань, завершених за певний період.

4. **Середній WIP:**

Обчислює середню кількість завдань, що перебувають у процесі виконання за весь проєкт.

5. **Utilization Rate (завантаження ресурсів):**

Вимірює ефективність використання доступних ресурсів.

2.2.4 Приклад реалізації Kanban у проєкті FinancePlus

У рамках проекту FinancePlus модель Kanban була реалізована з використанням бібліотеки simpy. Беклог містив завдання з різною тривалістю, такі як розробка інтерфейсу, створення функцій для додавання транзакцій і інтеграція з банківськими API.

1. **WIP-ліміт:**

Встановлено ліміт у 3 завдання для роботи в процесі, щоб уникнути перевантаження.

2. **Симуляція:**

Завдання виконувалися паралельно, а час виконання розраховувався з

урахуванням випадкових відхилень (±20%).

3. Результати:

- Усього було виконано 26 завдань.
- Загальний час виконання проєкту склав X годин.
- Метрики показали стабільну пропускну здатність і високий рівень завантаження ресурсів.

```
[0] Задача 'Розробка архітектури проєкту' взята у работу (Оценка: 20 ч, Фактическое: 17 ч)
[0] Задача 'Дизайн основного інтерфейсу' взята у работу (Оценка: 32 ч, Фактическое: 35 ч)
[0] Задача 'Розробка іконок та візуальних елементів' взята у работу (Оценка: 24 ч, Фактическое: 24 ч)
[17] Задача 'Розробка архітектури проєкту' виконана. Defect Density: 0.4, Customer Satisfaction: 66.69%, Actual Duration: 17 ч
[17] Задача 'Дизайн графіків та діаграм' взята у работу (Оценка: 24 ч, Фактическое: 22 ч)
[52] Задача 'Дизайн основного інтерфейсу' виконана. Defect Density: 0.96, Customer Satisfaction: 75.98%, Actual Duration: 35 ч
[52] Задача 'Дизайн повідомлень та повідомлень' взята у работу (Оценка: 20 ч, Фактическое: 21 ч)
[76] Задача 'Розробка іконок та візуальних елементів' виконана. Defect Density: 0.4, Customer Satisfaction: 83.68%, Actual Duration: 24 ч
[76] Задача 'Розробка головного меню та налаштувань' взята у работу (Оценка: 40 ч, Фактическое: 44 ч)
[98] Задача 'Дизайн графіків та діаграм' виконана. Defect Density: 0.47, Customer Satisfaction: 92.64%, Actual Duration: 22 ч
[98] Задача 'Розробка екрану додавання транзакцій' взята у работу (Оценка: 40 ч, Фактическое: 36 ч)
[119] Задача 'Дизайн повідомлень та повідомлень' виконана. Defect Density: 0.58, Customer Satisfaction: 66.76%, Actual Duration: 21 ч
[119] Задача 'Створення функції додавання транзакцій' взята у работу (Оценка: 40 ч, Фактическое: 47 ч)
[163] Задача 'Розробка головного меню та налаштувань' виконана. Defect Density: 0.9, Customer Satisfaction: 79.76%, Actual Duration: 44 ч
[163] Задача 'Категоризація транзакцій' взята у работу (Оценка: 24 ч, Фактическое: 25 ч)
[199] Задача 'Розробка екрану додавання транзакцій' виконана. Defect Density: 0.83, Customer Satisfaction: 73.94%, Actual Duration: 36 ч
[199] Задача 'Збереження даних користувача' взята у работу (Оценка: 20 ч, Фактическое: 22 ч)
[246] Задача 'Створення функції додавання транзакцій' виконана. Defect Density: 0.25, Customer Satisfaction: 68.65%, Actual Duration: 47 ч
[246] Задача 'Побудова графіків витрат' взята у работу (Оценка: 40 ч, Фактическое: 39 ч)
[271] Задача 'Категоризація транзакцій' виконана. Defect Density: 0.51, Customer Satisfaction: 66.78%, Actual Duration: 25 ч
[271] Задача 'Автоматичне формування звітів' взята у работу (Оценка: 24 ч, Фактическое: 24 ч)
[293] Задача 'Збереження даних користувача' виконана. Defect Density: 0.66, Customer Satisfaction: 70.04%, Actual Duration: 22 ч
[293] Задача 'Система нагадувань про регулярні платежі' взята у работу (Оценка: 24 ч, Фактическое: 27 ч)
[332] Задача 'Побудова графіків витрат' виконана. Defect Density: 0.24, Customer Satisfaction: 90.26%, Actual Duration: 39 ч
[332] Задача 'Інтеграція з банківськими API' взята у работу (Оценка: 60 ч, Фактическое: 66 ч)
[356] Задача 'Автоматичне формування звітів' виконана. Defect Density: 0.41, Customer Satisfaction: 84.31%, Actual Duration: 24 ч
[356] Задача 'Реалізація цілей заощаджень' взята у работу (Оценка: 32 ч, Фактическое: 35 ч)
[383] Задача 'Система нагадувань про регулярні платежі' виконана. Defect Density: 0.37, Customer Satisfaction: 84.81%, Actual Duration: 27 ч
[383] Задача 'Система захисту та шифрування даних' взята у работу (Оценка: 20 ч, Фактическое: 18 ч)
[449] Задача 'Інтеграція з банківськими API' виконана. Defect Density: 0.35, Customer Satisfaction: 92.31%, Actual Duration: 66 ч
[449] Задача 'Налаштування багатофакторної автентифікації' взята у работу (Оценка: 24 ч, Фактическое: 25 ч)
[484] Задача 'Реалізація цілей заощаджень' виконана. Defect Density: 0.66, Customer Satisfaction: 91.5%, Actual Duration: 35 ч
[484] Задача 'Юніт-тестування основних функцій' взята у работу (Оценка: 40 ч, Фактическое: 33 ч)
[502] Задача 'Система захисту та шифрування даних' виконана. Defect Density: 0.41, Customer Satisfaction: 62.37%, Actual Duration: 18 ч
[502] Задача 'Інтеграційне тестування' взята у работу (Оценка: 40 ч, Фактическое: 47 ч)
[527] Задача 'Налаштування багатофакторної автентифікації' виконана. Defect Density: 0.73, Customer Satisfaction: 64.75%, Actual Duration: 25 ч
[527] Задача 'Стрес-тестування системи' взята у работу (Оценка: 40 ч, Фактическое: 37 ч)
[560] Задача 'Юніт-тестування основних функцій' виконана. Defect Density: 0.7, Customer Satisfaction: 92.61%, Actual Duration: 33 ч
```

```
[560] Задача 'Налагодження та оптимізація продуктивності' взята у работу (Оценка: 40 ч, Фактическое: 45 ч)
[607] Задача 'Інтеграційне тестування' виконана. Defect Density: 0.18, Customer Satisfaction: 69.14%, Actual Duration: 47 ч
[607] Задача 'Інтеграція Google API для синхронізації даних' взята у работу (Оценка: 16 ч, Фактическое: 16 ч)
[644] Задача 'Стрес-тестування системи' виконана. Defect Density: 0.17, Customer Satisfaction: 87.25%, Actual Duration: 37 ч
[644] Задача 'Інтеграція платіжних систем' взята у работу (Оценка: 16 ч, Фактическое: 17 ч)
[689] Задача 'Налагодження та оптимізація продуктивності' виконана. Defect Density: 0.45, Customer Satisfaction: 71.73%, Actual Duration: 45 ч
[689] Задача 'Налаштування аналітичних інструментів' взята у работу (Оценка: 32 ч, Фактическое: 27 ч)
[705] Задача 'Інтеграція Google API для синхронізації даних' виконана. Defect Density: 0.59, Customer Satisfaction: 71.13%, Actual Duration: 16 ч
[705] Задача 'Впровадження системи PUSH-повідомлень' взята у работу (Оценка: 24 ч, Фактическое: 26 ч)
[722] Задача 'Інтеграція платіжних систем' виконана. Defect Density: 0.24, Customer Satisfaction: 67.92%, Actual Duration: 17 ч
[722] Задача 'Додавання рейтингової системи користувачів' взята у работу (Оценка: 32 ч, Фактическое: 37 ч)
[749] Задача 'Налаштування аналітичних інструментів' виконана. Defect Density: 0.79, Customer Satisfaction: 77.57%, Actual Duration: 27 ч
[775] Задача 'Впровадження системи PUSH-повідомлень' виконана. Defect Density: 0.52, Customer Satisfaction: 98.91%, Actual Duration: 26 ч
[812] Задача 'Додавання рейтингової системи користувачів' виконана. Defect Density: 0.44, Customer Satisfaction: 94.09%, Actual Duration: 37 ч

Результати часу виконання збережено у файл results_time_Kanban.xlsx

Результати метрик збережено у файл results_metrics_Kanban.xlsx

Кількість виконаних завдань: 26

Загальний час, витрачений на проект Kanban: 812 годин
```

Рисунок 2.2 – результат запуску моделі Scrum

2.3 Імітаційна модель Waterfall

Waterfall-модель відображає класичний послідовний підхід до розробки програмного забезпечення. У цій моделі кожна фаза виконується окремо, і лише після її завершення починається наступна. У симуляції для проекту FinancePlus застосовуються механізми випадкових відхилень у часі, а також метрики для оцінки якості виконання фаз.

2.3.3 Основні елементи Waterfall

1. **Беклог:**

Кожна фаза має атрибути: назву, заплановану та фактичну тривалість, щільність дефектів та рівень задоволеності замовника.

2. **Робота в процесі:**

Завдання виконуються у фіксованій послідовності.

3. **Метрики:**

Defect Density (щільність дефектів).

Customer Satisfaction (задоволеність замовника).

4. **Ресурси:**

Середовище симуляції (simpy) для виконання фаз.

5. **Excel-звіти**

Час виконання фаз записується в один файл, а метрики — в інший.

2.3.4 Опис логіки моделі

1. **Додавання фаз проекту:**

Кожна фаза проекту додається у список фаз з відповідною оцінкою тривалості.

2. **Виконання фаз:**

- Тривалість варіюється випадковим чином (±20%), щоб моделювати реальні відхилення.
- Завершення однієї фази є обов’язковою умовою для початку наступної.

3. Метрики:

Для кожної фази генеруються:

- Defect Density: випадкове значення у діапазоні [0.1, 1.0].
- Customer Satisfaction: випадкове значення у діапазоні [60, 100] %.

4. Запис результатів:

Результат часу виконання та метрики записуються у два окремих файли Excel:

- result_time_Waterfall.xlsx: Містить назву фаз, оцінки тривалості та файктичний час виконання
- result_metrics_Waterfall.xlsx: Містить назви фаз, Defect Densite та Customer Setisfaction.

2.3.5 Метрики Waterfall

Для оцінки продуктивності моделі використовуються такі метрики:

1. Defect Density (щільність дефектів):

- Показує частоту дефектів, які виникли під час виконання фази.
- Випадкова величина моделює можливі ризики, пов’язані з якістю виконання.

За її допомогою можна розробити проміжні етапи тестування, щоб уникнути накопичення помилок. Наприклад, використовуючи формулу:

$$Defect\ Density = \frac{\text{Кількість дефектів}}{\text{Обсяг коду(у } LOC\ —\ Lines\ of\ Code, \text{рядки коду)}}$$

Якщо у проєкті виявлено 100 дефектів на 10,000 рядків коду, то густина

дефектів становить:

$$Defect\ Density = \frac{100}{10000} = 0.01 \text{ дефектів на рядок коду}$$

Зменшення цього показника можливе за рахунок регулярного тестування після кожного етапу розробки.

2. **Customer Satisfaction (задоволеність замовника):**

- Вимірює задоволеність замовника після завершення кожної фази.
- Випадкова величина у межах [60, 100] %.

3. **Фактична тривалість:**

Відображає реальний час, витрачений на виконання кожної фази.

4. **Загальний час:**

Сумарний час на виконання всіх фаз проекту.

2.3.6 Приклад реалізації Waterfall у проекті FinancePlus

5. **Вхідні дані:**

У проекті було додано 26 фаз із запланованою тривалістю.

6. **Реалізація:**

- Фази виконувалися послідовно.
- Тривалість фаз варіювалася в межах ±20% від запланованої.

7. **Результати:**

- Загальний час виконання проекту: X годин.
- Defect Density для фаз: $[0.4, 0.8, \dots]$.
- Customer Satisfaction для фаз: $[85, 92, \dots]$.
- Усі метрики та час виконання записані у відповідні Excel-файли.

```
[0] Початок фази: Розробка архітектури проєкту (Фактичний час: 18 годин)
[18] Фаза 'Розробка архітектури проєкту' завершена. Defect Density: 0.3, Customer Satisfaction: 94.33%, Defects: 0
[18] Початок фази: Дизайн основного інтерфейсу (Фактичний час: 28 годин)
[46] Фаза 'Дизайн основного інтерфейсу' завершена. Defect Density: 0.85, Customer Satisfaction: 98.69%, Defects: 0
[46] Початок фази: Розробка іконок та візуальних елементів (Фактичний час: 28 годин)
[74] Фаза 'Розробка іконок та візуальних елементів' завершена. Defect Density: 0.86, Customer Satisfaction: 73.75%, Defects: 0
[74] Початок фази: Дизайн графіків і діаграм (Фактичний час: 26 годин)
[100] Фаза 'Дизайн графіків і діаграм' завершена. Defect Density: 0.68, Customer Satisfaction: 68.98%, Defects: 0
[100] Початок фази: Дизайн сповіщень та повідомлень (Фактичний час: 21 годин)
[121] Фаза 'Дизайн сповіщень та повідомлень' завершена. Defect Density: 0.77, Customer Satisfaction: 63.56%, Defects: 0
[121] Початок фази: Розробка головного меню та налаштувань (Фактичний час: 46 годин)
[167] Фаза 'Розробка головного меню та налаштувань' завершена. Defect Density: 0.88, Customer Satisfaction: 70.65%, Defects: 0
[167] Початок фази: Розробка екрану додавання транзакцій (Фактичний час: 36 годин)
[203] Фаза 'Розробка екрану додавання транзакцій' завершена. Defect Density: 0.33, Customer Satisfaction: 84.25%, Defects: 0
[203] Початок фази: Створення функції додавання транзакцій (Фактичний час: 34 годин)
[237] Фаза 'Створення функції додавання транзакцій' завершена. Defect Density: 0.4, Customer Satisfaction: 87.66%, Defects: 0
[237] Початок фази: Категоризація транзакцій (Фактичний час: 25 годин)
[262] Фаза 'Категоризація транзакцій' завершена. Defect Density: 0.33, Customer Satisfaction: 80.65%, Defects: 0
[262] Початок фази: Збереження даних користувача (Фактичний час: 18 годин)
[280] Фаза 'Збереження даних користувача' завершена. Defect Density: 0.18, Customer Satisfaction: 69.46%, Defects: 3
[280] Початок фази: Побудова графіків витрат (Фактичний час: 47 годин)
[327] Фаза 'Побудова графіків витрат' завершена. Defect Density: 0.62, Customer Satisfaction: 84.56%, Defects: 0
[327] Початок фази: Автоматичне формування звітів (Фактичний час: 20 годин)
[347] Фаза 'Автоматичне формування звітів' завершена. Defect Density: 0.58, Customer Satisfaction: 89.54%, Defects: 0
[347] Початок фази: Система нагадувань про регулярні платежі (Фактичний час: 21 годин)
[368] Фаза 'Система нагадувань про регулярні платежі' завершена. Defect Density: 0.41, Customer Satisfaction: 86.0%, Defects: 0
[368] Початок фази: Інтеграція з банківськими API (Фактичний час: 57 годин)
[425] Фаза 'Інтеграція з банківськими API' завершена. Defect Density: 0.37, Customer Satisfaction: 66.98%, Defects: 0
[425] Початок фази: Реалізація цілей збережень (Фактичний час: 34 годин)
[459] Фаза 'Реалізація цілей збережень' завершена. Defect Density: 0.35, Customer Satisfaction: 76.79%, Defects: 0
[459] Початок фази: Система захисту та шифрування даних (Фактичний час: 23 годин)
[482] Фаза 'Система захисту та шифрування даних' завершена. Defect Density: 0.73, Customer Satisfaction: 61.72%, Defects: 0
[482] Початок фази: Налаштування багатофакторної автентифікації (Фактичний час: 21 годин)
[503] Фаза 'Налаштування багатофакторної автентифікації' завершена. Defect Density: 0.12, Customer Satisfaction: 78.21%, Defects: 0
[503] Початок фази: Юніт-тестування основних функцій (Фактичний час: 47 годин)
[550] Фаза 'Юніт-тестування основних функцій' завершена. Defect Density: 0.99, Customer Satisfaction: 64.56%, Defects: 0
[550] Початок фази: Інтеграційне тестування (Фактичний час: 41 годин)
[591] Фаза 'Інтеграційне тестування' завершена. Defect Density: 0.19, Customer Satisfaction: 91.71%, Defects: 0
[591] Початок фази: Стрес-тестування системи (Фактичний час: 37 годин)
[628] Фаза 'Стрес-тестування системи' завершена. Defect Density: 0.69, Customer Satisfaction: 85.38%, Defects: 0
[628] Початок фази: Налагодження та оптимізація продуктивності (Фактичний час: 43 годин)
[671] Фаза 'Налагодження та оптимізація продуктивності' завершена. Defect Density: 0.58, Customer Satisfaction: 80.03%, Defects: 0
[671] Початок фази: Інтеграція Google API для синхронізації даних (Фактичний час: 15 годин)
[686] Фаза 'Інтеграція Google API для синхронізації даних' завершена. Defect Density: 0.46, Customer Satisfaction: 82.4%, Defects: 0
[686] Початок фази: Інтеграція платіжних систем (Фактичний час: 16 годин)
[702] Фаза 'Інтеграція платіжних систем' завершена. Defect Density: 0.1, Customer Satisfaction: 61.34%, Defects: 0
[702] Початок фази: Налаштування аналітичних інструментів (Фактичний час: 30 годин)
[732] Фаза 'Налаштування аналітичних інструментів' завершена. Defect Density: 0.26, Customer Satisfaction: 67.17%, Defects: 0
[732] Початок фази: Впровадження системи PUSH-повідомлень (Фактичний час: 29 годин)
[761] Фаза 'Впровадження системи PUSH-повідомлень' завершена. Defect Density: 0.37, Customer Satisfaction: 72.42%, Defects: 0
[761] Початок фази: Додавання рейтингової системи користувачів (Фактичний час: 31 годин)
[792] Фаза 'Додавання рейтингової системи користувачів' завершена. Defect Density: 0.12, Customer Satisfaction: 81.2%, Defects: 0

Результати часу виконання збережено у файл results_time_Waterfall.xlsx

Результати метрик збережено у файл results_metrics_Waterfall.xlsx

Кількість виконаних фаз: 26

Загальний час, витрачений на проект Waterfall: 792 годин
```

Рисунок 2.3 – результат запуску моделі Waterfall

8. Підсумок:

- У моделі було змодельовано 26 фаз із різними рівнями якості.
- Отримані метрики дали змогу оцінити ключові ризики проекту.

2.3.7 Порівняння з іншими моделями

1. **Гнучкість:**

Waterfall жорстко фіксує порядок виконання фаз, що може бути менш адаптивним порівняно зі Scrum.

2. **Метрики:**

Додаються специфічні показники, такі як задоволеність замовника, які не включені в Scrum.

3. **Складність:**

Реалізація Waterfall простіша через чітку структуру.

Висновок до розділу 2

У цьому розділі було розроблено та описано моделі для трьох популярних методологій розробки програмного забезпечення: Scrum, Kanban і Waterfall. Кожна з цих моделей була детально розглянута, створена за допомогою симуляційного середовища SimPy та інтегрована з інструментами для збору і збереження метрик у форматі Excel.

Особливу увагу приділено структурі та логіці кожної моделі, включаючи визначення основних компонентів, таких як завдання, фази чи етапи проекту, а також метрик, таких як тривалість виконання завдань, густина дефектів і задоволеність клієнтів.

Для досягнення максимального наближення моделей до реальних умов були використані наступні підходи:

1. **Випадкові відхилення у тривалості завдань**: У моделях передбачено можливість випадкових відхилень у тривалості виконання завдань у межах ±20%. Це враховує непередбачувані обставини, які можуть вплинути на час виконання в реальних проектах.
2. **Ймовірність виникнення дефектів**: Дефекти в завданнях генеруються з урахуванням рівня покриття тестами. Чим менше покриття, тим вища ймовірність виникнення дефектів. Це відображає реальний вплив якості тестування на стабільність програмного забезпечення.
3. **Генерація метрик задоволеності клієнтів**: Задоволеність клієнтів обчислюється на основі густини дефектів, що дає змогу симулювати реальний вплив якості продукту на сприйняття користувачами.

Ці підходи були обрані через їхню здатність адекватно моделювати невизначеність та змінність, притаманну реальним проектам. Таким чином, моделі дозволяють більш точно симулювати процеси розробки, що дає змогу отримати дані, наближені до реальних умов.

Таким чином, другий розділ забезпечив базу для наступного етапу дослідження, який полягатиме в аналізі ефективності кожної методології на основі зібраних даних. Застосування моделей, які враховують реалістичні умови, дозволяє зробити висновки більш обґрунтованими та практично значущими.

Розділ 3 Аналіз результатів застосування підходів Scrum, Kanban та Waterfall за допомогою метрик

У сучасному світі розробки програмного забезпечення вибір підходу до управління проектами є важливим фактором, який впливає на ефективність команд, якість продукту та задоволення клієнтів. Проаналізуємо три популярні підходи — Scrum, Kanban і Waterfall — на основі наданих даних, враховуючи основні показники: тривалість виконання завдань, Defect Density (густина дефектів), Customer Satisfaction (задоволеність клієнтів) і ефективність використання ресурсів.

3.1 Проект «FinancePlus»: Список задач

Був вигаданий додаток «FinancePlus» та задачі для подальшого моделювання у методологіях Scrum, Kanban та Waterfall на основі цього проекту.

Список задач на термін всього проєкту «FinancePlus» (Project Backlog):

Дизайн (220 год):

- Розробка архітектури проєкту (20 год)
- Дизайн основного інтерфейсу (32 год)
- Розробка іконок та візуальних елементів (24 год)
- Дизайн графіків і діаграм (24 год)
- Дизайн сповіщень та повідомлень (20 год)
- Розробка головного меню та налаштувань (40 год)
- Розробка екрану додавання транзакцій (40 год)
- Створення функції додавання транзакцій (20 год)

Розробка функціоналу (480 год):

- Категоризація транзакцій (24 год)

- Збереження даних користувача (20 год)
- Побудова графіків витрат (40 год)
- Автоматичне формування звітів (24 год)
- Система нагадувань про регулярні платежі (24 год)
- Інтеграція з банківськими API (60 год)
- Реалізація цілей збережень (32 год)
- Система захисту та шифрування даних (20 год)
- Налаштування багатофакторної автентифікації (24 год)

Тестування та налагодження (160 год):

- Юніт-тестування основних функцій (40 год)
- Інтеграційне тестування (40 год)
- Стрес-тестування системи (40 год)
- Налагодження та оптимізація продуктивності (40 год)

Інтеграція сторонніх сервісів (120 год):

- Інтеграція Google API для синхронізації даних (16 год)
- Інтеграція платіжних систем (16 год)
- Налаштування аналітичних інструментів (32 год)
- Впровадження системи PUSH-повідомлень (24 год)
- Додавання рейтингової системи користувачів (32 год)

Загальний час виконання: 960 годин

3.1 Scrum

3.1.1 Загальні результати

- **Загальний час виконання проекту:** 793,5 годин**.**

770	764	810	773	804	774	794	815	818	813	793,5

Рисунок 3.1 – результат середнього загального часу виконання проекту за 10 запусків моделей

- **Середня густина дефектів (Defect Density):** 0.15**.**

0,013701923	0,015064103	0,009615385	0,022596154	0,032211538	0,007371795	0,008333333	0,01698718	0,006410256	0,023878205	0,01561699

Рисунок 3.2 – результат середньої густини дефектів за 10 запусків моделей

- **Середній рівень задоволення клієнта (Customer Satisfaction):** 94.43%**.**

98,62980769	98,49358974	99,03846154	97,74038462	96,77884615	99,26282051	99,16666667	98,30128205	99,35897436	97,61217949	98,4383013

Рисунок 3.3 – результат середнього рівня задоволеності клієнта за 10 запусків моделей

3.1.2 Ключові спостереження

1. **Ефективність часу виконання**: Scrum добре справляється з керуванням часом, що виявляється у тому, що завдання розподілені на спринти, що допомагає уникати значних затримок. Однак для завдань з високим рівнем складності, таких як "Побудова графіків витрат" (45 годин) та "Інтеграція з банківськими API" (67 годин), спостерігаються значні часові витрати.
2. **Щільність дефектів:** Метрика залишається помірною (0.15), що говорить про середні показники якості коду. Scrum допомагає мінімізувати помилки завдяки регулярним ретроспективам та спринтовим перевіркам.
3. **Задоволеність клієнта:** На рівні 94,43% Scrum демонструє високий результат. Клієнти отримують оновлення після кожного спринту, що збільшує їхню залученість та задоволення.

3.1.3 Переваги Scrum за даними

- Гнучкість: дозволяє реагувати зміни вимог у процесі роботи.

- Регулярні перевірки якості за рахунок спринтів та ретроспектив.
- Високий рівень комунікації усередині команди.

3.1.4 Обмеження Scrum

- Нерівномірний розподіл часу на завдання: складні завдання можуть виходити за межі запланованих спринтів.
- Завищені часові витрати на комунікацію та планування.

3.2 Kanban

3.2.1 Загальні результати

- **Общее время выполнения проекта:** 796,8 часа.

804	823	829	787	761	769	795	812	818	770	796,8

Рисунок 3.4 – результат середнього загального часу виконання проекту за 10 запусків моделей

- **Средняя плотность дефектов (Defect Density):** 0.569.

0,537308	0,581154	0,665	0,628462	0,557308	0,546923	0,510769	0,559231	0,54	0,564615	0,569077

Рисунок 3.5 – результат середньої густини дефектів за 10 запусків моделей

- **Средний уровень удовлетворённости клиента (Customer Satisfaction):** 80.26%.

78,78423	81,33308	79,21577	82,40692	80,18269	79,63115	79,00423	75,99538	82,91385	83,145	80,26123

Рисунок 3.6 – результат середнього рівня задоволеності клієнта за 10 запусків моделей

3.2.2 Ключові спостереження

1. **Ефективність часу виконання**: Kanban показав найбільш збалансований час виконання завдань. Система "pull" дозволяє розпочинати роботу лише тоді, коли ресурси доступні, що знижує

навантаження команди. Це особливо видно на задачах "Збереження даних користувача" (15 год) та "Реалізація цілей заощаджень" (27 год).

2. **Щільність дефектів:** Середнє значення (0.569) вказує трохи більш високу частоту дефектів проти Scrum. Відсутність фіксованих етапів ревью може бути причиною такого результату.
3. **Задоволеність клієнта:** Задоволеність клієнтів нижче, ніж у Scrum, через відсутність чітких проміжних результатів, які регулярно надаються замовнику

3.2.3 Переваги Kanban за даними

- Простота впровадження: не потребує радикальних змін у процесі.
- Усунення перевантажень команди завдяки гнучкому управлінню завданнями.
- Хороша візуалізація прогресу за допомогою дошки Kanban.

3.2.4 Обмеження Kanban

- Складнощі з пріоритетами: при високому обсязі завдань фокус на найважливіші завдання може бути втрачено.
- Відсутність чітких часових рамок для завершення завдань може збільшити термін виконання.

3.3 Waterfall

3.3.1 Загальні результати

- **Загальний час виконання проекту:** 792,8 години.

740	799	824	813	798	817	777	787	790	783	792,8

Рисунок 3.7 – результат середнього загального часу виконання проекту за 10 запусків моделей

- **Середня густина дефектів (Defect Density):** 0.554.

0,518076923	0,498076923	0,605	0,603846154	0,553076923	0,505384615	0,490384615	0,516153846	0,614615385	0,638076923	0,554269231

Рисунок 3.8 – результат середньої густини дефектів за 10 запусків моделей

- **Середній рівень задоволеності клієнта (Customer Satisfaction):** 80.14%**.**

79,96384615	78,92192308	74,57538462	80,75038462	78,44153846	81,71153846	82,81192308	81,74923077	81,34730769	81,17384615	80,14469231

Рисунок 3.9 – результат середнього рівня задоволеності клієнта за 10 запусків моделей

3.3.2 Ключові спостереження

1. **Ефективність часу виконання:** Waterfall демонструє найвищий час виконання завдань. Це з фіксованими етапами роботи, де кожна стадія має бути завершено до початку наступної. Наприклад, завдання "Категоризація транзакцій" (28 год) та "Автоматичне формування звітів" (29 год) зайняли більше часу, ніж в інших підходах.
2. **Щільність дефектів:** Метрика дефектів найвища серед підходів (0.554). Жорстка структура процесу не дозволяє швидко реагувати на зміни або знаходити дефекти на ранніх стадіях.
3. **Задоволеність клієнта:** Задоволеність клієнта нижче в порівнянні з Scrum та Kanban через відсутність регулярної взаємодії з клієнтом протягом усього циклу розробки**.**

3.3.3 Переваги Waterfall за даними

- Чітка структура: кожен етап має чіткий початок і кінець, що дозволяє краще контролювати процес.
- Підходить для проектів із жорсткими вимогами та низькою ймовірністю змін.

3.3.4 Обмеження Waterfall

- Низька гнучкість: труднощі щодо змін вимог на пізніх стадіях.

- Висока ймовірність дефектів, оскільки тестування проводиться лише наприкінці.
- Довгі цикли виконання завдань призводять до зниження задоволеності клієнтів.

3.4 Глобальне порівняння методик

Результати моделювання показали, що кожен із підходів має свої унікальні особливості, що впливають на ефективність виконання проєкту. Для кращого розуміння побудовано порівняльну таблицю ключових показників.

Таблиця 3.1 – порівняння ключових показників методологій

Метрики	Scrum	Kanban	Waterfall
Середня тривалість задач	Низька	Середня	Висока
Щільність дефектів	Середня	Низька	Найнижча
Задоволеність клієнта(%)	Висока	Висока	Низька
Гнучкість	Висока	Найвища	Низька
Придатність до планування	Середня	Низька	Найвища

Для визначення числових діапазонів позначень "Низька", "Середня", і "Висока" у таблиці можна орієнтуватися на загальний розподіл значень ключових метрик:

1. Середня тривалість задач (у годинах):
    - Низька: до 20 годин.
    - Середня: від 21 до 40 годин.
    - Висока: понад 40 годин.
2. Густина дефектів (Defect Density) (дефекти на одиницю часу чи коду):
    - Низька: до 0.3.
    - Середня: від 0.31 до 0.6.
    - Висока: понад 0.6.
3. Задоволеність клієнтів (у відсотках):

- Низька: до 70%.
- Середня: від 71% до 85%.
- Висока: понад 85%.

4. Гнучкість і придатність до планування (якісні оцінки, за шкалою від 1 до 10):

- Низька: 1–3 бали.
- Середня: 4–7 балів.
- Висока: 8–10 балів.

Таблиця 3.2 – порівняння середніх значень за 10 запусків моделей

Метрики	Scrum	Kanban	Waterfall
Загальній час (год)	793.5	797.8	792.8
Щільність дефектів	0.15	0.569	0.554
Задоволеність клієнта(%)	94.43	80.26	80.14

Аналіз з таблиці:

- Scrum найбільш ефективний у плані якості та задоволеності клієнта, але час виконання трохи вищий, ніж у Kanban.
- Kanban демонструє мінімальний час виконання, але якість результатів страждає через вищу щільність дефектів.
- Waterfall підходить для проектів з фіксованими вимогами, але показує найгірші результати за всіма ключовими метриками.

Висновок до розділу 3

У третьому розділі було проведено комплексний аналіз результатів застосування методологій Scrum, Kanban та Waterfall у контексті проекту "FinancePlus". Кожна методологія була оцінена за кількома ключовими метриками, що включали тривалість виконання завдань, щільність дефектів, задоволеність клієнтів і ефективність використання ресурсів.

Методологія Scrum виявилася ефективною для проектів, що потребують високої якості та регулярного зворотного зв'язку з клієнтами. Ітеративний підхід Scrum дозволив швидко виявляти дефекти і адаптуватися до змін вимог, що забезпечило високу якість кінцевого продукту. Однак, цей підхід вимагав більше часу на виконання проекту порівняно з іншими методологіями.

Kanban продемонстрував мінімальний час виконання завдань завдяки безперервному потоку роботи і обмеженню кількості одночасно виконуваних завдань (WIP). Це дозволило оптимізувати використання ресурсів і зменшити час на завершення проекту. Однак, через відсутність чітко структурованих етапів перевірки, щільність дефектів була дещо вищою, що може вплинути на якість кінцевого продукту.

Waterfall, з його лінійним підходом, забезпечив передбачуваність і структурованість, що є корисним для проектів із чітко визначеними вимогами. Проте, його жорстка послідовність етапів і відсутність гнучкості стали причиною накопичення дефектів, які виявлялися лише на кінцевих етапах тестування. Це зробило Waterfall менш придатним для проектів, де вимоги можуть змінюватися в процесі розробки.

На основі порівняльного аналізу можна зробити висновок, що вибір оптимальної методології значною мірою залежить від специфіки проекту, його вимог до якості, часу і ресурсів. Важливим є також врахування культури команди та її здатності до адаптації. Застосування адаптивного підходу до вибору методології дозволяє підвищити ефективність проектного управління

і зменшити ризики, пов'язані з розробкою програмного забезпечення.

Розділ 4 Дослідження залежності розміру команди на продуктивність у методоліях

4.1 Виявлення залежності розміру команди для моделі

У цьому розділі буде здійснено практичне застосування створених моделей для дослідження впливу різних методологій розробки програмного забезпечення на результати проєкту. Проводимиметься поглиблений аналіз отриманих результатів та здійснюватиметься оцінка практичної застосовності моделей. Наша мета — виявити, які з моделей найбільше відповідають реаліям розробки та які методології можна вважати найбільш ефективними в певних умовах.

4.1.1 Спільні формули

Складність виконання завдання в залежності від розміру команди в моделі розрахована за допомогою часу на комунікацію. Основний принцип полягає в тому, що зі збільшенням кількості учасників у команді зростає потреба у взаємодії між ними. Це призводить до збільшення часу, необхідного для обговорень, узгоджень і координації.

Час виконання одиниці роботи

Час виконання завдання або фази враховує базову тривалість, множник складності (за потреби) та час на комунікацію:

$$T_{total} = T_{base} * duration\_multiplier + T_{comm}$$

де:

T_{base} – базова тривалість одиниці роботи (завдання чи фази);

$T_{comm} = 0.05 * \big(N * (N-1)\big)$ – час на комунікацію, який залежить від розміру команди N.

duration_multiplier – множник складності (використовується у Scrum та Kanban):

- Проста: duration_multiplier = 0.8
- Середня: duration_multiplier = 1.0,
- Складна: duration_multiplier = 1.5.

Час на комунікацію

Час на комунікацію визначається кількістю пар у команді (Scrum, Waterfall, Kanban):

$$T_{comm} = F * (N * (N - 1)),$$

де:

F = 0.05 – коефіцієнт часу на комунікацію для кожної при членів команди (у годинах);

N – кількість учасників команди.

Продуктивність команди

Продуктивність команди оцінюється через кількість виконаних одиниць роботи за одиницю часу (Scrum, Waterfall):

$$P = \frac{V}{T},$$

P – продуктивність (виконані задачі за sодиницю часу);

V – обсяг виконання задач;

T – загальний час виконання.

Відхилення фактичного часу виконання

Для імітації реалістичних умов базовий час виконання модифікується на ±20% (Scrum, Waterfall):

$$T_{base}^{actual} = T_{base} \pm (T_{base} * 0.2)$$

4.1.2 Формули для Scrum

Кількість задач у спринті

Кількість задач у спринті визначається залежно від розміру команди:

$$Tasks\_per\_sprint = 2 * N$$

де N – кількість учасників команди

Загальний час виконання спринту

Сумарний час виконання у межах одного спринту:

$$T_{sprint} = \sum_{i=1}^{n} (T_{base,i} * duration_{multiplier} + T_{comm}),$$

де n – кількість задач у спринтію

4.1.3 Формули для Kanban

Ліміт задач у роботі (WIP)

Кількість задач, які можуть одночасно перебувати у виконанні:

$$WIP = fixed\_limit$$

де fixed_limit здається я фіксоване значення (наприклад, 5).

Час виконання потоку завдань

Сумарний час виконання всіх задач у потоці:

$$T_{sprint} = \sum_{i=1}^{n} (T_{base,i} * duration_{multiplier} + T_{comm}),$$

де n – загальна кількість завдань.

4.1.4 Формули для Waterfall

Тривалість виконання завдання

Формула для розрахунку фактичного часу виконання завдання враховує відхилення від базового часу та час на комунікацію (Scrum, Waterfall):

$$T_{total} = T_{base} + T_{comm},$$

де:

T_{total} – загальний час виконання завдання;

T_{base} – базовий час на виконання завдання;

T_{comm} – час на комунікацію між членами команди.

Формула для загального обсягу роботи

У моделі ми забезпечуємо однаковий обсяг роботи:

$$W_{total} = V_{tasks} * T_{task} = \sum_{i=0}^{n} T_{phase},$$

де:

W_{total} – загальний обсяг роботи (у годинах);

V_{tasks} – кількість задач у Scrum

T_{task} – середня тривалість у Scrum

T_{phase} – тривалість кожної фази у Waterfall

4.1.5 Аналіз результатів залежностей розміру команд для моделей

Аналіз результатів із 50 запусків для кожної моделі дозволило зробити кілька висновків про ефективність методологій залежно від розміру команди. Було простестовано на чотирьох розмірах команди, а саме команди розмірами 3, 5, 7, 10 виконавців.

Таблиця 4.1 – відношення кількості спринтів до розміру команди

	Команда 3 ч.	Команда 5 ч.	Команда 7 ч.	Команда 10ч.
Спринт 1	6	10	14	20
Спринт 2	12	20	28	30
Спринт 3	18	30	30	-
Спринт 4	24	-	-	-

Спринт 5	30	-	-	-

Scrum

Збільшення розміру команди в моделі Scrum значно впливає на загальний час виконання проєкту, демонструючи чітку тенденцію зростання. У невеликих командах, що складаються з 3-5 осіб, загальний час виконання суттєво менший порівняно з командами, які включають 10 і більше учасників. Основна причина цього явища полягає в тому, що зі збільшенням розміру команди ускладнюється комунікація між її учасниками. Кожен додатковий член команди створює нові канали взаємодії, які потребують більше часу на координацію, обговорення задач та вирішення конфліктів.

Для невеликих команд менша кількість учасників забезпечує простіші комунікаційні канали, що дозволяє ефективніше координувати зусилля, уникати тривалих обговорень та швидше приймати рішення. У великих командах цей процес стає складнішим, адже час, витрачений на обмін інформацією, обговорення та досягнення консенсусу, суттєво зростає. Таким чином, оптимальний розмір команди для моделі Scrum є важливим фактором, який варто враховувати при плануванні проєкту, щоб досягти максимальної ефективності та мінімізувати витрати часу.

Таблиця 4.2 – середній результат запуску моделі Scrum

Розмір команди	Загальний час (у годинах)
3	275.66
5	292.72
7	321.92
10	395.24

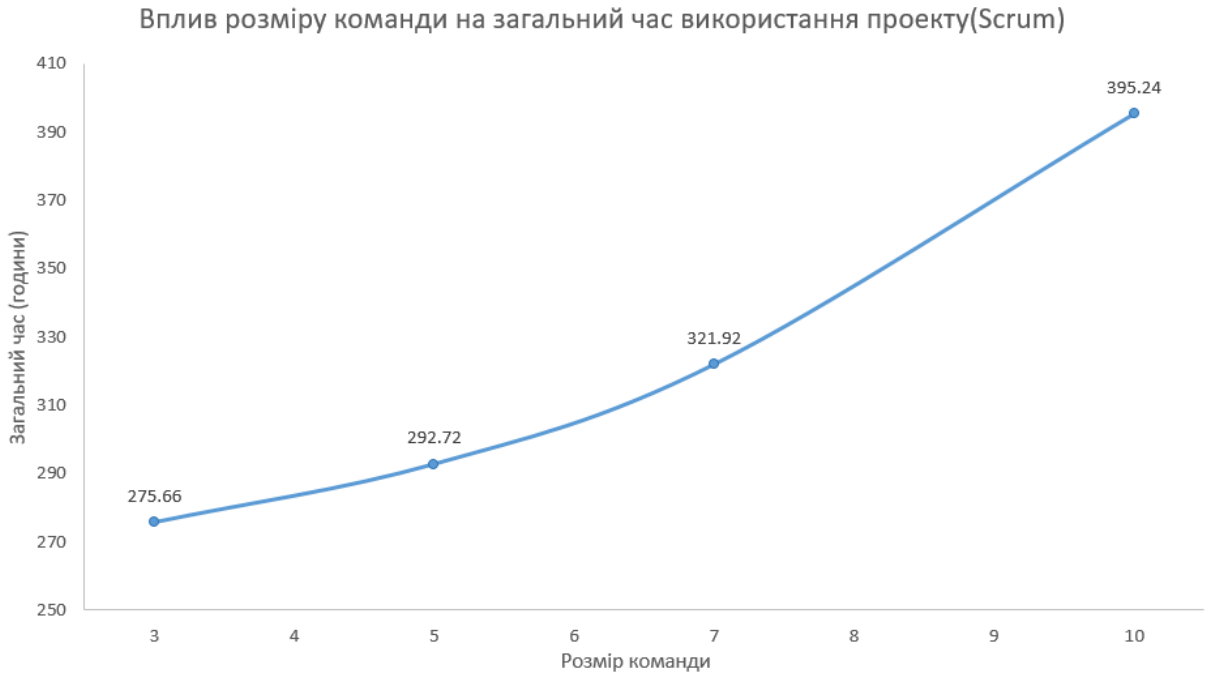

Рисунок 4.1 – графік впливу команди на загальний час виконання проекту

Kanban

Результати моделювання для Kanban показують майже ідентичні значення порівняно зі Scrum, із різницею, що не перевищує 1-2% і може бути врахована в межах статистичної похибки. Це вказує на те, що обидві методології демонструють схожу ефективність за однакових умов.

Основною причиною цього є те, що Kanban, як і Scrum, враховує час, необхідний для комунікації між членами команди. Проте його ключова перевага — гнучкість і прозорість процесів — дозволяє компенсувати вплив великого розміру команди на продуктивність. Завдяки безперервному потоку роботи і можливості швидкого реагування на зміни, Kanban демонструє високу стійкість до складнощів, пов'язаних із масштабуванням команд, що робить його ефективним навіть у великих колективах.

Таблиця 4.3 – середній результат запуску моделі Kanban

Розмір команди	Загальний час (у годинах)
3	268.38
5	290.58

7	321.58
10	391.22

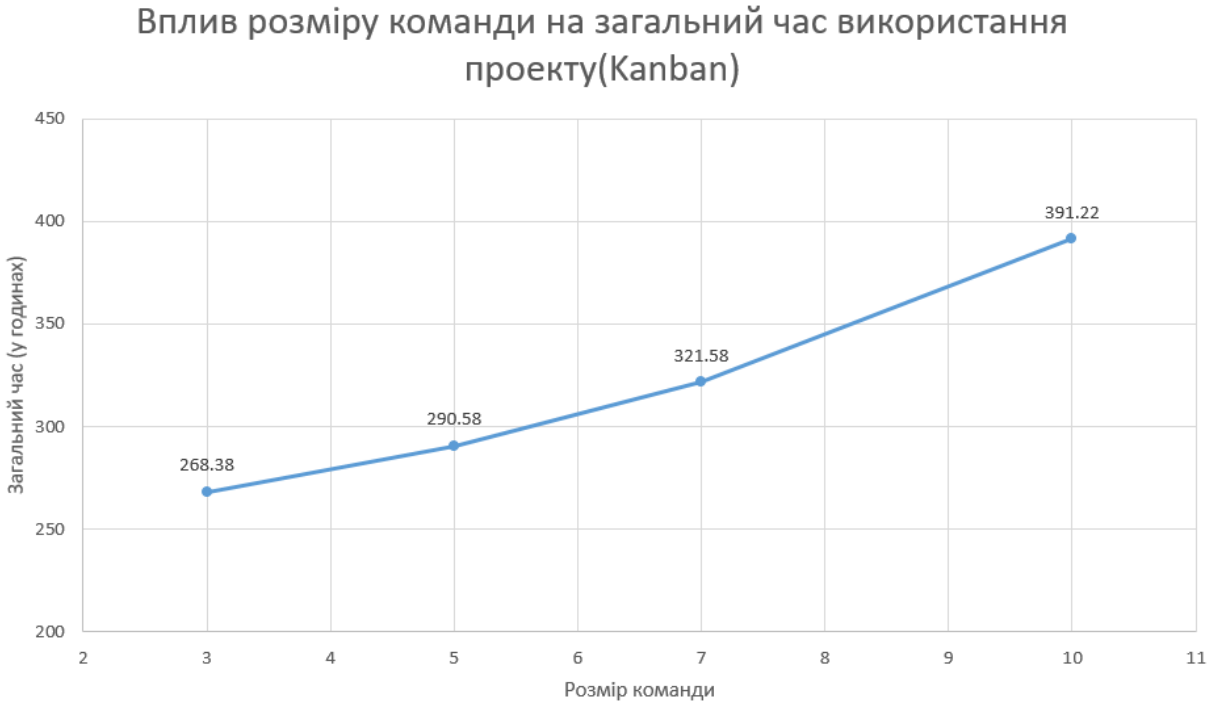

Рис 4.2 – графік впливу команди на загальний час виконання проекту

Waterfall

У моделі Waterfall розмір команди має значно менший вплив на загальний час виконання проєкту порівняно зі Scrum і Kanban. Результати моделювання показують, що час виконання практично не змінюється при збільшенні кількості учасників команди з 3 до 7 осіб, а відмінності залишаються мінімальними. Лише при залученні 10 членів команди спостерігається помітне зростання тривалості виконання проєкту.

Ця особливість пояснюється стабільною структурою роботи, притаманною Waterfall. У цій моделі кожна фаза виконується послідовно, і всі завдання розподіляються між членами команди заздалегідь. Такий підхід мінімізує необхідність постійної комунікації та координації, характерної для гнучких методологій. Завдяки цьому збільшення кількості учасників не створює суттєвого навантаження на процеси планування чи комунікації. Waterfall

залишається передбачуваним і ефективним навіть для команд із різним розміром, хоча його продуктивність трохи знижується у великих командах через потребу в більшій кількості комунікацій у складних фазах.

Таблиця 4.4 – середній результат запуску моделі Kanban

Розмір команди	Загальний час (у годинах)
3	339.89
5	346.34
7	346.46
10	366.36

Рисунок 4.3 – графік впливу команди на загальний час виконання проекту

Висновок до розділу 4

У цьому розділі було проведено глибокий аналіз впливу розміру команди на продуктивність виконання проектів у контексті трьох основних методологій розробки програмного забезпечення: Scrum, Kanban і Waterfall. Використовуючи симуляційні моделі, були розглянуті різні розміри команд і їхній вплив на ключові показники продуктивності, такі як швидкість виконання завдань, кількість дефектів і загальна ефективність проекту.

Для методології Scrum встановлено, що існує оптимальний розмір команди, при якому продуктивність досягає максимального рівня. Після досягнення цього оптимуму подальше збільшення команди призводить до зниження ефективності через складнощі в координації і комунікації. Це підкреслює важливість дотримання рекомендованого розміру команди в рамках Scrum.

Kanban показав більшу стійкість до змін у розмірі команди, оскільки ця методологія менше залежить від ітераційної структури та більше орієнтована на постійний потік завдань. Це дозволяє командам з різним числом учасників підтримувати високу продуктивність, особливо якщо обсяг незавершеної роботи (WIP) контролюється належним чином.

Методологія Waterfall продемонструвала відносно незалежність продуктивності від розміру команди, що пов'язано з її послідовною природою, де кожен етап розробки виконується окремо, з чітким розподілом завдань. Однак цей підхід також має свої обмеження, особливо в умовах змінних вимог і необхідності швидкої адаптації.

Отримані результати свідчать про те, що вибір оптимального розміру команди повинен враховувати особливості методології, проекту та конкретні завдання. Це дозволяє підвищити ефективність роботи, зменшити кількість дефектів і забезпечити кращу якість кінцевого продукту. Впровадження цих висновків у практику управління проектами може сприяти підвищенню

конкурентоспроможності команди та зниженню ризиків, пов'язаних з розробкою програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України "Про вищу освіту" [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1556-18

2. Войтенко, І. О. Методи оцінки продуктивності розробницьких команд у середовищах Agile та Waterfall / І. О. Войтенко. – Харків: ХНУРЕ, 2021. – 320 с.

3. SimPy Documentation: Simulation framework for Python [Електронний ресурс]. – Режим доступу: https://simpy.readthedocs.io

4. Шевченко, О. С. Алгоритми обробки метрик у програмних системах управління проєктами / О. С. Шевченко, В. П. Іванов. – Харків: ХНУРЕ, 2018. – 290 с.

5. Excel Documentation: Офіційний довідник по роботі з Excel API [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/office

Висновки

У ході виконання даної магістерської роботи було проведено ґрунтовне дослідження процедур збору та обробки даних життєвого циклу розробки програмного забезпечення (ЖЦРПЗ). Аналіз різних методологій, таких як Scrum, Kanban та Waterfall, дозволив оцінити їхні особливості, сильні та слабкі сторони, а також вплив на продуктивність та якість роботи команд розробників.

Основні результати дослідження підтвердили, що кожна з методологій має свої унікальні переваги у певних контекстах. Зокрема, Scrum забезпечує гнучкість і швидке реагування на зміни, що є важливим у динамічних проектах. Kanban виявився ефективним у проектах, що потребують постійного вдосконалення процесів. Водночас, Waterfall забезпечує високу передбачуваність і структурованість, що є критично важливим для проектів з чітко визначеними вимогами. Також було досліджено вплив розміру команди на продуктивність виконання проектів у різних методологіях. Результати показали, що оптимальний розмір команди значно варіюється залежно від обраної методології. Наприклад, у Scrum менші команди часто виявляються більш ефективними, оскільки це сприяє кращій комунікації та швидкому прийняттю рішень. У випадку з Waterfall, більші команди можуть бути доцільними для одночасного виконання різних етапів розробки, хоча це може підвищити складність управління.

Одним із важливих аспектів роботи було моделювання реальних умов розробки програмного забезпечення, що дозволило оцінити ефективність різних підходів у реальних сценаріях. Використання сучасних інструментів та метрик для оцінки продуктивності команд сприяло підвищенню прозорості та контрольованості процесів розробки.

Новизна дослідження полягає у розробці та модифікації моделей, які враховують реалістичні умови та випадкові відхилення у тривалості завдань. Це дозволяє більш точно оцінювати ефективність команд і приймати

обґрунтовані рішення щодо вибору оптимальної методології для конкретних проектів.

Практична значущість роботи полягає у можливості застосування отриманих результатів для вдосконалення підходів до збору та обробки даних у програмній інженерії. Це сприятиме підвищенню ефективності командної роботи, покращенню якості програмного забезпечення та зниженню ризиків, пов'язаних з управлінням проектами.

Рекомендації, отримані в результаті дослідження, включають оптимізацію вибору метрик для різних типів проектів, автоматизацію процесів збору даних і використання отриманих результатів для підтримки прийняття рішень. Подальші дослідження можуть бути зосереджені на глибшому аналізі інтеграції нових інструментів для збору даних у розробці ПЗ та їхнього впливу на продуктивність команд.

Таким чином, проведене дослідження робить вагомий внесок у розвиток практик управління проектами у сфері програмної інженерії, забезпечуючи надійні інструменти для підвищення ефективності та якості розробки програмного забезпечення.

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

АНАЛІЗ ПРОЦЕДУР ЗБОРУ ТА ОБРОБКИ ДАНИХ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01445-01-ЛЗ

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Вадим ГОРЯЧКІН

Виконавець

Георгій СІНЬКОВ

Нормоконтролер

Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01445-01-ЛЗ

АНАЛІЗ ПРОЦЕДУР ЗБОРУ ТА ОБРОБКИ ДАНИХ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технічне завдання

44165850.01445-01-ЛЗ

Листів 06

2025

АНОТАЦІЯ

Документ 44165850.01445-01 «Аналіз процедур збору та обробки даних життєвого циклу розробки програмного забезпечення» Технічне завдання, що входить до складу програмної документації до дипломного проекту.

Цей документ визначає мету та сферу застосування програми, встановлює основні вимоги до неї, надає інформацію про етапи та терміни виконання проекту. Крім того, в програмній документації розглядаються технічні аспекти програми, а також її техніко-економічні показники. Це включає в себе технічні характеристики, методи розробки, тести, аналіз ефективності та вартість реалізації проекту. Програмна документація до дипломного проекту включає в себе детальний опис та інформацію про розроблену програм.

ВСТУП

У сучасному світі програмне забезпечення стало невід'ємною частиною майже всіх сфер людської діяльності, від особистого використання до великих корпоративних систем. Ефективність розробки програмного забезпечення значною мірою залежить від обраної методології, правильного збору та обробки даних, що використовуються для аналізу і прийняття рішень під час життєвого циклу розробки. Саме тому виникає необхідність у розробці підходів, які дозволять підвищити продуктивність та якість програмного забезпечення шляхом оптимізації процесів збору даних та вибору найбільш підходящих методологій.

Ця дипломна робота спрямована на аналіз різних методологій розробки програмного забезпечення, таких як Scrum, Kanban та Waterfall, з метою оцінки їх впливу на продуктивність команд розробників. Дослідження також включає моделювання процесів розробки, використання сучасних інструментів для збору даних та оцінку впливу розміру команди на ефективність виконання проектів.

1 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ ректора Українського державного університету науки і технологій Радкевич А.В. “Про затвердження тем та призначення керівників дипломних проектів” №1196 ст від 05.12.2022 року.

Тема проекті: “Аналіз процедур збору та обробки даних життєвого циклу розробки програмного забезпечення”.

Керівник - доцент Горячкін В. М.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення цієї роботи полягає в аналізі та оптимізації процедур збору та обробки даних у життєвому циклі розробки програмного забезпечення (ЖЦРПЗ). Основною метою є дослідження методів, які дозволяють покращити якість програмних продуктів та підвищити ефективність роботи команд розробників. Це включає вибір метрик, автоматизацію процесів обробки даних, а також інтеграцію сучасних інструментів управління проектами.

Експлуатаційне призначення цієї роботи спрямоване на впровадження результатів дослідження у реальні умови розробки програмного забезпечення. Це передбачає застосування розроблених моделей і процедур для покращення управління проектами, підвищення продуктивності команд, а також для більш точного прогнозування термінів виконання проектів. Робота може служити основою для впровадження нових підходів у процеси розробки програмного забезпечення, забезпечуючи гнучкість і адаптивність до змін у вимогах та умовах ринку.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функіональниз характеристик

Підтримка моделювання різних методологій: Програма повинна забезпечувати моделювання процесів розробки програмного забезпечення для методологій Scrum, Kanban і Waterfall. Це включає можливість імітації завдань з урахуванням різних факторів, таких як розмір команди та випадкові відхилення у тривалості виконання завдань.

Збір і обробка даних: Програмне забезпечення повинно автоматично збирати дані про виконання завдань, включаючи тривалість виконання, щільність дефектів та інші релевантні метрики. Зібрані дані повинні бути доступні для подальшого аналізу.

Аналіз і візуалізація результатів: Програма повинна надавати засоби для аналізу зібраних даних та їх візуалізації у вигляді графіків і звітів. Це дозволить користувачам легко оцінити ефективність обраних методологій і зробити висновки про оптимальні стратегії для різних проектів.

Гнучкість налаштувань: Користувачі повинні мати можливість налаштовувати параметри моделювання, такі як кількість учасників команди, тривалість ітерацій та обмеження на кількість завдань у роботі (WIP).

Підтримка звітності: Програма повинна забезпечувати можливість створення звітів, що містять ключові показники ефективності проектів і результати моделювання.

3.2 Вимоги до надійності

Стійкість до помилок:

При запуску всі дані прописані, що виключає з’являння помилок при вводі даних.

3.3 Умови експлуатації

Сумісність:

Підтримка роботи на різних платформах та середовищах.

Інтерфейс:

Програма запускається як сприпт с записом результатів у файли.

Застосування:

Можливість використання для обробки збору та подальшого аналізу даних у різних випадках.

3.4 Вимоги до складу і параметрів технічних засобів

Мінімальні вимоги до обсягу оперативної пам’яті та просесорної потужності.

3.5 Вимоги до маркування і упаковки

Доступність документації, включаючи пояснення методів роботи та використання.

Забезпечення належної упаковки програмного продукту для зручного розгортання та використання.

3.6 Вимоги до транспортування і зберігання

Можливість зберігання результатів обчислення для подальшого використання.

Можливість перенесення програм між різними системами.

Додаток Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

АНАЛІЗ ПРОЦЕДУР ЗБОРУ ТА ОБРОБКИ ДАНИХ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Керівництво користувача

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01445-01 І3 01

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Вадим ГОРЯЧКІН

Виконавець

Георгій СІНЬКОВ

Нормоконтролер

Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01445-01 І3 01

АНАЛІЗ ПРОЦЕДУР ЗБОРУ ТА ОБРОБКИ ДАНИХ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Керівництво користувача

44165850.01445-01 І3 01

Листів 06

2025

АНОТАЦІЯ

Документ 44165850.01445-01 І3 01 «Аналіз процедур збору та обробки даних життєвого циклу розробки програмного забезпечення. Керівництво користувача»входить до складу програмної докуметації на програму, що реалізує алгоритми методик Scrum, Kanban та Waterfall.

У даному документі представлено керівництво користувача. Програма написана на мові Python.

Программа не потребує конкретної ОС, вимагає лише наявність PyCharm.

1. Введення

У сучасному світі програмне забезпечення стало невід'ємною частиною майже всіх сфер людської діяльності, від особистого використання до великих корпоративних систем. Ефективність розробки програмного забезпечення значною мірою залежить від обраної методології, правильного збору та обробки даних, що використовуються для аналізу і прийняття рішень під час життєвого циклу розробки. Саме тому виникає необхідність у розробці підходів, які дозволять підвищити продуктивність та якість програмного забезпечення шляхом оптимізації процесів збору даних та вибору найбільш підходящих методологій.

Це Керівництво користувача призначене для ознайомлення з програмним забезпеченням, розробленим в рамках дипломної роботи, яка досліджує ефективність різних методологій розробки програмного забезпечення. Програма була створена для демонстрації та аналізу впливу методологій Scrum, Kanban та Waterfall на продуктивність команд розробників.

2. Призначення та умови застосування

Програмне забезпечення, розроблене в рамках дипломної роботи, призначене для моделювання та аналізу різних методологій розробки програмного забезпечення, таких як Scrum, Kanban і Waterfall. Основною метою програми є надання інструментів для оцінки ефективності цих методологій шляхом збору та обробки даних, що дозволяє користувачам оптимізувати робочі процеси та підвищити продуктивність команд розробників.

Програма призначена для використання в середовищі розробників програмного забезпечення, проектних менеджерів та аналітиків, які прагнуть покращити процеси розробки через глибокий аналіз і моделювання.

Програмне забезпечення призначене для використання в наступних умовах:

- Командна розробка: Програма ефективно підтримує команди різного розміру, дозволяючи оцінювати вплив розміру команди на продуктивність у контексті різних методологій.
- Проекти з різними вимогами: Завдяки підтримці кількох методологій, програма може бути застосована до проектів із різними вимогами до гнучкості, термінів виконання та якості.
- Аналіз даних: Програма підходить для проектів, де важливим є збір та аналіз великих обсягів даних для прийняття обґрунтованих рішень щодо оптимізації процесів розробки.

Програма може бути використана як у навчальних, так і в професійних цілях, забезпечуючи користувачів ефективними засобами для вивчення та покращення процесів розробки програмного забезпечення.

3. Підготовка до роботи

Для запуску додатку користувачу необхідно мати Scrum_metric.py, Kanban_metric.py, Waterfall_metric.py, Scrum_size_team.py, Kanban_size_team.py, Waterfall_size_team.py.

Також необхідне встановленя PyCharm.

4. Опис операцій

Программа має декілька моделювань

```
=== Початок спринту 1 ===
[0] Виконується завдання: Розробка архітектури проєкту (Оцінка: 20 годин, Фактичний час: 23 годин)
[23] Завдання 'Розробка архітектури проєкту' завершено. Дефекти: 0, Густина дефектів: 0.00
[23] Виконується завдання: Дизайн основного інтерфейсу (Оцінка: 32 годин, Фактичний час: 37 годин)
[60] Завдання 'Дизайн основного інтерфейсу' завершено. Дефекти: 0, Густина дефектів: 0.00
[60] Виконується завдання: Розробка іконок та візуальних елементів (Оцінка: 24 годин, Фактичний час: 22 годин)
[82] Завдання 'Розробка іконок та візуальних елементів' завершено. Дефекти: 0, Густина дефектів: 0.00
[82] Виконується завдання: Дизайн графіків і діаграм (Оцінка: 24 годин, Фактичний час: 20 годин)
[102] Завдання 'Дизайн графіків і діаграм' завершено. Дефекти: 2, Густина дефектів: 0.08
[102] Виконується завдання: Дизайн сповіщень та повідомлень (Оцінка: 20 годин, Фактичний час: 23 годин)
[125] Завдання 'Дизайн сповіщень та повідомлень' завершено. Дефекти: 2, Густина дефектів: 0.10
=== Завершення спринту 1 ===

=== Початок спринту 2 ===
[125] Виконується завдання: Розробка головного меню та налаштувань (Оцінка: 40 годин, Фактичний час: 37 годин)
[162] Завдання 'Розробка головного меню та налаштувань' завершено. Дефекти: 3, Густина дефектів: 0.07
[162] Виконується завдання: Розробка екрану додавання транзакцій (Оцінка: 40 годин, Фактичний час: 48 годин)
[210] Завдання 'Розробка екрану додавання транзакцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[210] Виконується завдання: Створення функції додавання транзакцій (Оцінка: 40 годин, Фактичний час: 36 годин)
[246] Завдання 'Створення функції додавання транзакцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[246] Виконується завдання: Категоризація транзакцій (Оцінка: 24 годин, Фактичний час: 22 годин)
[268] Завдання 'Категоризація транзакцій' завершено. Дефекти: 2, Густина дефектів: 0.08
[268] Виконується завдання: Збереження даних користувача (Оцінка: 20 годин, Фактичний час: 18 годин)
[286] Завдання 'Збереження даних користувача' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 2 ===

=== Початок спринту 3 ===
[286] Виконується завдання: Побудова графіків витрат (Оцінка: 40 годин, Фактичний час: 41 годин)
[327] Завдання 'Побудова графіків витрат' завершено. Дефекти: 0, Густина дефектів: 0.00
[327] Виконується завдання: Автоматичне формування звітів (Оцінка: 24 годин, Фактичний час: 21 годин)
[348] Завдання 'Автоматичне формування звітів' завершено. Дефекти: 0, Густина дефектів: 0.00
[348] Виконується завдання: Система нагадувань про регулярні платежі (Оцінка: 24 годин, Фактичний час: 20 годин)
[368] Завдання 'Система нагадувань про регулярні платежі' завершено. Дефекти: 3, Густина дефектів: 0.12
[368] Виконується завдання: Інтеграція з банківськими API (Оцінка: 60 годин, Фактичний час: 70 годин)
[438] Завдання 'Інтеграція з банківськими API' завершено. Дефекти: 0, Густина дефектів: 0.00
[438] Виконується завдання: Реалізація цілей збережень (Оцінка: 32 годин, Фактичний час: 27 годин)
[465] Завдання 'Реалізація цілей збережень' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 3 ===
```

```
=== Початок спринту 4 ===
[465] Виконується завдання: Система захисту та шифрування даних (Оцінка: 20 годин, Фактичний час: 21 годин)
[486] Завдання 'Система захисту та шифрування даних' завершено. Дефекти: 0, Густина дефектів: 0.00
[486] Виконується завдання: Налаштування багатофакторної автентифікації (Оцінка: 24 годин, Фактичний час: 29 годин)
[515] Завдання 'Налаштування багатофакторної автентифікації' завершено. Дефекти: 1, Густина дефектів: 0.04
[515] Виконується завдання: Юніт-тестування основних функцій (Оцінка: 40 годин, Фактичний час: 36 годин)
[551] Завдання 'Юніт-тестування основних функцій' завершено. Дефекти: 0, Густина дефектів: 0.00
[551] Виконується завдання: Інтеграційне тестування (Оцінка: 40 годин, Фактичний час: 47 годин)
[598] Завдання 'Інтеграційне тестування' завершено. Дефекти: 0, Густина дефектів: 0.00
[598] Виконується завдання: Стрес-тестування системи (Оцінка: 40 годин, Фактичний час: 37 годин)
[635] Завдання 'Стрес-тестування системи' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 4 ===

=== Початок спринту 5 ===
[635] Виконується завдання: Налагодження та оптимізація продуктивності (Оцінка: 40 годин, Фактичний час: 39 годин)
[674] Завдання 'Налагодження та оптимізація продуктивності' завершено. Дефекти: 0, Густина дефектів: 0.00
[674] Виконується завдання: Інтеграція Google API для синхронізації даних (Оцінка: 16 годин, Фактичний час: 16 годин)
[690] Завдання 'Інтеграція Google API для синхронізації даних' завершено. Дефекти: 2, Густина дефектів: 0.12
[690] Виконується завдання: Інтеграція платіжних систем (Оцінка: 16 годин, Фактичний час: 14 годин)
[704] Завдання 'Інтеграція платіжних систем' завершено. Дефекти: 0, Густина дефектів: 0.00
[704] Виконується завдання: Налаштування аналітичних інструментів (Оцінка: 32 годин, Фактичний час: 34 годин)
[738] Завдання 'Налаштування аналітичних інструментів' завершено. Дефекти: 0, Густина дефектів: 0.00
[738] Виконується завдання: Впровадження системи PUSH-повідомлень (Оцінка: 24 годин, Фактичний час: 21 годин)
[759] Завдання 'Впровадження системи PUSH-повідомлень' завершено. Дефекти: 0, Густина дефектів: 0.00
=== Завершення спринту 5 ===

=== Початок спринту 6 ===
[759] Виконується завдання: Додавання рейтингової системи користувачів (Оцінка: 32 годин, Фактичний час: 38 годин)
[797] Завдання 'Додавання рейтингової системи користувачів' завершено. Дефекти: 0, Густина дефектів: 0.00
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
Жодних завдань для виконання у цьому спринті.
=== Завершення спринту 6 ===
```

```
Результати часу виконання збережено у файл results_time_Scrum.xlsx

Результати метрик збережено у файл results_metrics_Scrum.xlsx

Кількість виконаних завдань: 26

Загальний час, витрачений на проект Scrum: 797 годин
```

Моделювання методології Scrum

```
[0] Задача 'Розробка архітектури проєкту' взята у работу (Оценка: 20 ч, Фактическое: 17 ч)
[0] Задача 'Дизайн основного інтерфейсу' взята у работу (Оценка: 32 ч, Фактическое: 35 ч)
[0] Задача 'Розробка іконок та візуальних елементів' взята у работу (Оценка: 24 ч, Фактическое: 24 ч)
[17] Задача 'Розробка архітектури проєкту' виконана. Defect Density: 0.4, Customer Satisfaction: 66.69%, Actual Duration: 17 ч
[17] Задача 'Дизайн графіків та діаграм' взята у работу (Оценка: 24 ч, Фактическое: 22 ч)
[52] Задача 'Дизайн основного інтерфейсу' виконана. Defect Density: 0.96, Customer Satisfaction: 75.98%, Actual Duration: 35 ч
[52] Задача 'Дизайн повідомлень та повідомлень' взята у работу (Оценка: 20 ч, Фактическое: 21 ч)
[76] Задача 'Розробка іконок та візуальних елементів' виконана. Defect Density: 0.4, Customer Satisfaction: 83.68%, Actual Duration: 24 ч
[76] Задача 'Розробка головного меню та налаштувань' взята у работу (Оценка: 40 ч, Фактическое: 44 ч)
[98] Задача 'Дизайн графіків та діаграм' виконана. Defect Density: 0.47, Customer Satisfaction: 92.64%, Actual Duration: 22 ч
[98] Задача 'Розробка екрану додавання транзакцій' взята у работу (Оценка: 40 ч, Фактическое: 36 ч)
[119] Задача 'Дизайн повідомлень та повідомлень' виконана. Defect Density: 0.58, Customer Satisfaction: 66.76%, Actual Duration: 21 ч
[119] Задача 'Створення функції додавання транзакцій' взята у работу (Оценка: 40 ч, Фактическое: 47 ч)
[163] Задача 'Розробка головного меню та налаштувань' виконана. Defect Density: 0.9, Customer Satisfaction: 79.76%, Actual Duration: 44 ч
[163] Задача 'Категоризація транзакцій' взята у работу (Оценка: 24 ч, Фактическое: 25 ч)
[199] Задача 'Розробка екрану додавання транзакцій' виконана. Defect Density: 0.83, Customer Satisfaction: 73.94%, Actual Duration: 36 ч
[199] Задача 'Збереження даних користувача' взята у работу (Оценка: 20 ч, Фактическое: 22 ч)
[246] Задача 'Створення функції додавання транзакцій' виконана. Defect Density: 0.25, Customer Satisfaction: 68.65%, Actual Duration: 47 ч
[246] Задача 'Побудова графіків витрат' взята у работу (Оценка: 40 ч, Фактическое: 39 ч)
[271] Задача 'Категоризація транзакцій' виконана. Defect Density: 0.51, Customer Satisfaction: 66.78%, Actual Duration: 25 ч
[271] Задача 'Автоматичне формування звітів' взята у работу (Оценка: 24 ч, Фактическое: 24 ч)
[293] Задача 'Збереження даних користувача' виконана. Defect Density: 0.66, Customer Satisfaction: 70.04%, Actual Duration: 22 ч
[293] Задача 'Система нагадувань про регулярні платежі' взята у работу (Оценка: 24 ч, Фактическое: 27 ч)
[332] Задача 'Побудова графіків витрат' виконана. Defect Density: 0.24, Customer Satisfaction: 90.26%, Actual Duration: 39 ч
[332] Задача 'Інтеграція з банківськими API' взята у работу (Оценка: 60 ч, Фактическое: 66 ч)
[356] Задача 'Автоматичне формування звітів' виконана. Defect Density: 0.41, Customer Satisfaction: 84.31%, Actual Duration: 24 ч
[356] Задача 'Реалізація цілей заощаджень' взята у работу (Оценка: 32 ч, Фактическое: 35 ч)
[383] Задача 'Система нагадувань про регулярні платежі' виконана. Defect Density: 0.37, Customer Satisfaction: 84.81%, Actual Duration: 27 ч
[383] Задача 'Система захисту та шифрування даних' взята у работу (Оценка: 20 ч, Фактическое: 18 ч)
[449] Задача 'Інтеграція з банківськими API' виконана. Defect Density: 0.35, Customer Satisfaction: 92.31%, Actual Duration: 66 ч
[449] Задача 'Налаштування багатофакторної автентифікації' взята у работу (Оценка: 24 ч, Фактическое: 25 ч)
[484] Задача 'Реалізація цілей заощаджень' виконана. Defect Density: 0.66, Customer Satisfaction: 91.5%, Actual Duration: 35 ч
[484] Задача 'Юніт-тестування основних функцій' взята у работу (Оценка: 40 ч, Фактическое: 33 ч)
[502] Задача 'Система захисту та шифрування даних' виконана. Defect Density: 0.41, Customer Satisfaction: 62.37%, Actual Duration: 18 ч
[502] Задача 'Інтеграційне тестування' взята у работу (Оценка: 40 ч, Фактическое: 47 ч)
[527] Задача 'Налаштування багатофакторної автентифікації' виконана. Defect Density: 0.73, Customer Satisfaction: 64.75%, Actual Duration: 25 ч
[527] Задача 'Стрес-тестування системи' взята у работу (Оценка: 40 ч, Фактическое: 37 ч)
[560] Задача 'Юніт-тестування основних функцій' виконана. Defect Density: 0.7, Customer Satisfaction: 92.61%, Actual Duration: 33 ч
```

```
[560] Задача 'Налагодження та оптимізація продуктивності' взята у работу (Оценка: 40 ч, Фактическое: 45 ч)
[607] Задача 'Інтеграційне тестування' виконана. Defect Density: 0.18, Customer Satisfaction: 69.14%, Actual Duration: 47 ч
[607] Задача 'Інтеграція Google API для синхронізації даних' взята у работу (Оценка: 16 ч, Фактическое: 16 ч)
[644] Задача 'Стрес-тестування системи' виконана. Defect Density: 0.17, Customer Satisfaction: 87.25%, Actual Duration: 37 ч
[644] Задача 'Інтеграція платіжних систем' взята у работу (Оценка: 16 ч, Фактическое: 17 ч)
[689] Задача 'Налагодження та оптимізація продуктивності' виконана. Defect Density: 0.45, Customer Satisfaction: 71.73%, Actual Duration: 45 ч
[689] Задача 'Налаштування аналітичних інструментів' взята у работу (Оценка: 32 ч, Фактическое: 27 ч)
[705] Задача 'Інтеграція Google API для синхронізації даних' виконана. Defect Density: 0.59, Customer Satisfaction: 71.13%, Actual Duration: 16 ч
[705] Задача 'Впровадження системи PUSH-повідомлень' взята у работу (Оценка: 24 ч, Фактическое: 26 ч)
[722] Задача 'Інтеграція платіжних систем' виконана. Defect Density: 0.24, Customer Satisfaction: 67.92%, Actual Duration: 17 ч
[722] Задача 'Додавання рейтингової системи користувачів' взята у работу (Оценка: 32 ч, Фактическое: 37 ч)
[749] Задача 'Налаштування аналітичних інструментів' виконана. Defect Density: 0.79, Customer Satisfaction: 77.57%, Actual Duration: 27 ч
[775] Задача 'Впровадження системи PUSH-повідомлень' виконана. Defect Density: 0.52, Customer Satisfaction: 98.91%, Actual Duration: 26 ч
[812] Задача 'Додавання рейтингової системи користувачів' виконана. Defect Density: 0.44, Customer Satisfaction: 94.09%, Actual Duration: 37 ч

Результати часу виконання збережено у файл results_time_Kanban.xlsx

Результати метрик збережено у файл results_metrics_Kanban.xlsx

Кількість виконаних завдань: 26

Загальний час, витрачений на проект Kanban: 812 годин
```

Моделювання методології Kanban

```
[0] Початок фази: Розробка архітектури проєкту (Фактичний час: 18 годин)
[18] Фаза 'Розробка архітектури проєкту' завершена. Defect Density: 0.3, Customer Satisfaction: 94.33%, Defects: 0
[18] Початок фази: Дизайн основного інтерфейсу (Фактичний час: 28 годин)
[46] Фаза 'Дизайн основного інтерфейсу' завершена. Defect Density: 0.85, Customer Satisfaction: 98.69%, Defects: 0
[46] Початок фази: Розробка іконок та візуальних елементів (Фактичний час: 28 годин)
[74] Фаза 'Розробка іконок та візуальних елементів' завершена. Defect Density: 0.86, Customer Satisfaction: 73.75%, Defects: 0
[74] Початок фази: Дизайн графіків і діаграм (Фактичний час: 26 годин)
[100] Фаза 'Дизайн графіків і діаграм' завершена. Defect Density: 0.68, Customer Satisfaction: 68.98%, Defects: 0
[100] Початок фази: Дизайн сповіщень та повідомлень (Фактичний час: 21 годин)
[121] Фаза 'Дизайн сповіщень та повідомлень' завершена. Defect Density: 0.77, Customer Satisfaction: 63.56%, Defects: 0
[121] Початок фази: Розробка головного меню та налаштувань (Фактичний час: 46 годин)
[167] Фаза 'Розробка головного меню та налаштувань' завершена. Defect Density: 0.88, Customer Satisfaction: 70.65%, Defects: 0
[167] Початок фази: Розробка екрану додавання транзакцій (Фактичний час: 36 годин)
[203] Фаза 'Розробка екрану додавання транзакцій' завершена. Defect Density: 0.33, Customer Satisfaction: 84.25%, Defects: 0
[203] Початок фази: Створення функції додавання транзакцій (Фактичний час: 34 годин)
[237] Фаза 'Створення функції додавання транзакцій' завершена. Defect Density: 0.4, Customer Satisfaction: 87.66%, Defects: 0
[237] Початок фази: Категоризація транзакцій (Фактичний час: 25 годин)
[262] Фаза 'Категоризація транзакцій' завершена. Defect Density: 0.33, Customer Satisfaction: 80.65%, Defects: 0
[262] Початок фази: Збереження даних користувача (Фактичний час: 18 годин)
[280] Фаза 'Збереження даних користувача' завершена. Defect Density: 0.18, Customer Satisfaction: 69.46%, Defects: 3
[280] Початок фази: Побудова графіків витрат (Фактичний час: 47 годин)
[327] Фаза 'Побудова графіків витрат' завершена. Defect Density: 0.62, Customer Satisfaction: 84.56%, Defects: 0
[327] Початок фази: Автоматичне формування звітів (Фактичний час: 20 годин)
[347] Фаза 'Автоматичне формування звітів' завершена. Defect Density: 0.58, Customer Satisfaction: 89.54%, Defects: 0
[347] Початок фази: Система нагадувань про регулярні платежі (Фактичний час: 21 годин)
[368] Фаза 'Система нагадувань про регулярні платежі' завершена. Defect Density: 0.41, Customer Satisfaction: 86.0%, Defects: 0
[368] Початок фази: Інтеграція з банківськими API (Фактичний час: 57 годин)
[425] Фаза 'Інтеграція з банківськими API' завершена. Defect Density: 0.37, Customer Satisfaction: 66.98%, Defects: 0
[425] Початок фази: Реалізація цілей збережень (Фактичний час: 34 годин)
[459] Фаза 'Реалізація цілей збережень' завершена. Defect Density: 0.35, Customer Satisfaction: 76.79%, Defects: 0
[459] Початок фази: Система захисту та шифрування даних (Фактичний час: 23 годин)
[482] Фаза 'Система захисту та шифрування даних' завершена. Defect Density: 0.73, Customer Satisfaction: 61.72%, Defects: 0
[482] Початок фази: Налаштування багатофакторної автентифікації (Фактичний час: 21 годин)
[503] Фаза 'Налаштування багатофакторної автентифікації' завершена. Defect Density: 0.12, Customer Satisfaction: 78.21%, Defects: 0
[503] Початок фази: Юніт-тестування основних функцій (Фактичний час: 47 годин)
[550] Фаза 'Юніт-тестування основних функцій' завершена. Defect Density: 0.99, Customer Satisfaction: 64.56%, Defects: 0
[550] Початок фази: Інтеграційне тестування (Фактичний час: 41 годин)
[591] Фаза 'Інтеграційне тестування' завершена. Defect Density: 0.19, Customer Satisfaction: 91.71%, Defects: 0
```

```
[591] Початок фази: Стрес-тестування системи (Фактичний час: 37 годин)
[628] Фаза 'Стрес-тестування системи' завершена. Defect Density: 0.69, Customer Satisfaction: 85.38%, Defects: 0
[628] Початок фази: Налагодження та оптимізація продуктивності (Фактичний час: 43 годин)
[671] Фаза 'Налагодження та оптимізація продуктивності' завершена. Defect Density: 0.58, Customer Satisfaction: 80.03%, Defects: 0
[671] Початок фази: Інтеграція Google API для синхронізації даних (Фактичний час: 15 годин)
[686] Фаза 'Інтеграція Google API для синхронізації даних' завершена. Defect Density: 0.46, Customer Satisfaction: 82.4%, Defects: 0
[686] Початок фази: Інтеграція платіжних систем (Фактичний час: 16 годин)
[702] Фаза 'Інтеграція платіжних систем' завершена. Defect Density: 0.1, Customer Satisfaction: 61.34%, Defects: 0
[702] Початок фази: Налаштування аналітичних інструментів (Фактичний час: 30 годин)
[732] Фаза 'Налаштування аналітичних інструментів' завершена. Defect Density: 0.26, Customer Satisfaction: 67.17%, Defects: 0
[732] Початок фази: Впровадження системи PUSH-повідомлень (Фактичний час: 29 годин)
[761] Фаза 'Впровадження системи PUSH-повідомлень' завершена. Defect Density: 0.37, Customer Satisfaction: 72.42%, Defects: 0
[761] Початок фази: Додавання рейтингової системи користувачів (Фактичний час: 31 годин)
[792] Фаза 'Додавання рейтингової системи користувачів' завершена. Defect Density: 0.12, Customer Satisfaction: 81.2%, Defects: 0

Результати часу виконання збережено у файл results_time_Waterfall.xlsx

Результати метрик збережено у файл results_metrics_Waterfall.xlsx

Кількість виконаних фаз: 26

Загальний час, витрачений на проект Waterfall: 792 годин
```

Моделювання методології Watrrfall

Додаток В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського державного університету науки і технологій

Анатолій РАДКЕВИЧ

18.02.25

АНАЛІЗ ПРОЦЕДУР ЗБОРУ ТА ОБРОБКИ ДАНИХ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01445-01 12 01

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Керівник розробки

Вадим ГОРЯЧКІН

Виконавець

Георгій СІНЬКОВ

Нормоконтролер

Світлана ВОЛКОВА

2025

Код программи для Дослідження і моделювання даних житевого циклу розробки програмного забезпечення.

Scrum_metric

```
import simpy
import random
from openpyxl import load_workbook, Workbook
import os

# Клас для завдань (Tasks)
class Task:
    def __init__(self, name, duration, test_coverage):
        self.name = name
        self.duration = duration
        self.test_coverage = test_coverage  # Покриття коду тестами (у відсотках)
        self.start_time = None  # Час початку виконання завдання
        self.end_time = None  # Час завершення завдання
        self.defects = 0  # Кількість дефектів у завданні
        self.defect_density = 0  # Щільність дефектів
        self.customer_satisfaction = 0  # Задоволеність клієнта

# Клас для проекту FinancePlus
class FinancePlusProject:
    def __init__(self, env, tasks_per_sprint):
        self.env = env
        self.tasks_per_sprint = tasks_per_sprint
        self.backlog = []  # Список усіх завдань проєкту
        self.completed_tasks = []  # Список виконаних завдань
        self.total_time_spent = 0
        self.velocity = []  # Швидкість виконання завдань зі спринтів
        self.defect_density = []  # Щільність дефектів по спринтам
        self.task_defect_density = []  # Щільність дефектів по завданнях

    def add_task(self, name, duration, test_coverage):
        task = Task(name, duration, test_coverage)
        self.backlog.append(task)

    def simulate_task_duration(self, base_duration):
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))  # Мінімум 1 година

    def simulate_defects(self, task):
        defect_probability = (100 - task.test_coverage) / 100
        if random.random() < defect_probability:
            task.defects = random.randint(1, 3)

    def run_project_until_complete(self):
        sprint = 1
        while self.backlog:
            print(f"\n=== Початок спринту {sprint} ===")
            sprint_velocity = 0
```

```
        sprint_defects = 0

        for i in range(self.tasks_per_sprint):
            if self.backlog:
                task = self.backlog.pop(0)
                task.start_time = self.env.now

                actual_duration = self.simulate_task_duration(task.duration)
                print(f"[{self.env.now}] Виконується завдання: {task.name} (Оцінка: {task.duration} годин, Фактичний час: {actual_duration} годин)")

                self.total_time_spent += actual_duration
                yield self.env.timeout(actual_duration)

                task.end_time = self.env.now
                self.simulate_defects(task)

                task.defect_density = task.defects / task.duration if task.duration > 0 else 0
                task.customer_satisfaction = max(0, 100 - task.defect_density * 100)

                self.completed_tasks.append(task)
                sprint_velocity += 1
                sprint_defects += task.defects

                # Обчислення густини дефектів для завдання
                self.task_defect_density.append(task.defect_density)

                print(f"[{self.env.now}] Завдання '{task.name}' завершено. Дефекти: {task.defects}, Густина дефектів: {task.defect_density:.2f}")
            else:
                break

        self.velocity.append(sprint_velocity)
        self.defect_density.append(sprint_defects / sprint_velocity if sprint_velocity > 0 else 0)
        print(f"=== Завершення спринту {sprint} ===")
        sprint += 1

    def save_time_to_excel(self, file_name="results_time_Scrum.xlsx"):
        if os.path.exists(file_name):
            workbook = load_workbook(file_name)
        else:
            workbook = Workbook()
            workbook.active.title = "Time Results"

        sheet = workbook.active
        if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
            sheet.cell(row=1, column=1, value="Назва завдання")
            sheet.cell(row=1, column=2, value="Оцінка тривалості (год)")
            sheet.cell(row=1, column=3, value="Фактичний час (год)")

        for i, task in enumerate(self.completed_tasks, start=2):
            sheet.cell(row=i, column=1, value=task.name)
```

```
            sheet.cell(row=i, column=2, value=task.duration)
            sheet.cell(row=i, column=3, value=task.end_time - task.start_time)

        workbook.save(file_name)
        print(f"\nРезультати часу виконання збережено у файл {file_name}")

    def save_metrics_to_excel(self, file_name="results_metrics_Scrum.xlsx"):
        if os.path.exists(file_name):
            workbook = load_workbook(file_name)
        else:
            workbook = Workbook()
            workbook.active.title = "Metrics"

        sheet = workbook.active
        if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
            sheet.cell(row=1, column=1, value="Назва завдання")
            sheet.cell(row=1, column=2, value="Defect Density")
            sheet.cell(row=1, column=3, value="Customer Satisfaction (%)")

        for i, task in enumerate(self.completed_tasks, start=2):
            sheet.cell(row=i, column=1, value=task.name)
            sheet.cell(row=i, column=2, value=task.defect_density)
            sheet.cell(row=i, column=3, value=task.customer_satisfaction)

        workbook.save(file_name)
        print(f"\nРезультати метрик збережено у файл {file_name}")

# Ініціалізація середовища SimPy
env = simpy.Environment()

# Створення проекту FinancePlus
tasks_per_sprint = 5
project = FinancePlusProject(env, tasks_per_sprint)

# Додаємо завдання в беклог проекту FinancePlus
project.add_task("Розробка архітектури проєкту", 20, 80)
project.add_task("Дизайн основного інтерфейсу", 32, 70)
project.add_task("Розробка іконок та візуальних елементів", 24, 90)
project.add_task("Дизайн графіків і діаграм", 24, 85)
project.add_task("Дизайн сповіщень та повідомлень", 20, 75)
project.add_task("Розробка головного меню та налаштувань", 40, 80)
project.add_task("Розробка екрану додавання транзакцій", 40, 60)
project.add_task("Створення функції додавання транзакцій", 40, 50)
project.add_task("Категоризація транзакцій", 24, 90)
project.add_task("Збереження даних користувача", 20, 95)
project.add_task("Побудова графіків витрат", 40, 85)
project.add_task("Автоматичне формування звітів", 24, 88)
project.add_task("Система нагадувань про регулярні платежі", 24, 80)
project.add_task("Інтеграція з банківськими API", 60, 50)
project.add_task("Реалізація цілей збережень", 32, 75)
project.add_task("Система захисту та шифрування даних", 20, 85)
project.add_task("Налаштування багатофакторної автентифікації", 24, 80)
```

```
project.add_task("Юніт-тестування основних функцій", 40, 90)
project.add_task("Інтеграційне тестування", 40, 85)
project.add_task("Стрес-тестування системи", 40, 80)
project.add_task("Налагодження та оптимізація продуктивності", 40, 75)
project.add_task("Інтеграція Google API для синхронізації даних", 16, 90)
project.add_task("Інтеграція платіжних систем", 16, 85)
project.add_task("Налаштування аналітичних інструментів", 32, 80)
project.add_task("Впровадження системи PUSH-повідомлень", 24, 85)
project.add_task("Додавання рейтингової системи користувачів", 32, 80)

# Запускаємо симуляцію
env.process(project.run_project_until_complete())
env.run()

# Запис результатів у Excel
project.save_time_to_excel()
project.save_metrics_to_excel()

# Виведення загального часу
print(f"\nКількість виконаних завдань: {len(project.completed_tasks)}")
print(f"\nЗагальний час, витрачений на проект Scrum: {project.total_time_spent} годин")
```

Kanban_metric

```
import simpy
import random
from openpyxl import load_workbook, Workbook
import os

# Класс для задач (Tasks)
class Task:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
        self.actual_duration = 0
        self.defect_density = 0
        self.customer_satisfaction = 0

# Класс для модели Kanban
class KanbanProject:
    def __init__(self, env):
        self.env = env
        self.backlog = []  # Список усіх завдань
        self.in_progress = []  # Завдання у роботі
        self.completed = []  # Завершені завдання
        self.total_time_spent = 0

    # Додавання задачі в белог
    def add_task(self, name, duration):
        self.backlog.append(Task(name, duration))

    # Симуляція виконання завдання з урахуванням випадкового відхилення часу
    def simulate_task_duration(self, base_duration):
```

```
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))  # Мінімум 1 час

    # Генерація метрик для задачі
    def generate_task_metrics(self, task):
        task.defect_density = round(random.uniform(0.1, 1.0), 2)  # Дефектність
        task.customer_satisfaction = round(random.uniform(60, 100), 2)  # Задоволеність

    # Основна логіка роботи Kanban
    def run_project(self):
        while self.backlog or self.in_progress:
            # Перемещение задач из бэклога в работу
            while len(self.in_progress) < 3 and self.backlog:  # Обмеження на завдання у роботі
                task = self.backlog.pop(0)
                task.actual_duration = self.simulate_task_duration(task.duration)
                self.in_progress.append(task)
                print(f"[{self.env.now}] Задача '{task.name}' взята у работу (Оценка: {task.duration}
ч, Фактическое: {task.actual_duration} ч)")

            # Виконання завдань у роботі
            if self.in_progress:
                task = self.in_progress.pop(0)
                yield self.env.timeout(task.actual_duration)
                self.generate_task_metrics(task)
                self.completed.append(task)
                self.total_time_spent += task.actual_duration
                print(f"[{self.env.now}] Задача '{task.name}' виконана. Defect Density:
{task.defect_density}, Customer Satisfaction: {task.customer_satisfaction}%")

    # Запис часу виконання в Excel
    def save_time_to_excel(self, file_name="results_time_Kanban.xlsx"):
        if os.path.exists(file_name):
            workbook = load_workbook(file_name)
        else:
            workbook = Workbook()
            workbook.active.title = "Time Results"

        sheet = workbook.active

        # Встановлення заголовків під час першого запуску
        if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
            sheet.cell(row=1, column=1, value="Название задачи")
            sheet.cell(row=1, column=2, value="Оценка длительности (ч)")
            sheet.cell(row=1, column=3, value="Фактическое время (ч)")

        # Запис даних із завдань
        for i, task in enumerate(self.completed, start=2):
            sheet.cell(row=i, column=1, value=task.name)
            sheet.cell(row=i, column=2, value=task.duration)
            sheet.cell(row=i, column=3, value=task.actual_duration)
```

```
        workbook.save(file_name)
        print(f"\nРезультати часу виконання збережено у файл {file_name}")

    # Запис метрик у Excel
    def save_metrics_to_excel(self, file_name="results_metrics_Kanban.xlsx"):
        if os.path.exists(file_name):
            workbook = load_workbook(file_name)
        else:
            workbook = Workbook()
            workbook.active.title = "Metrics"

        sheet = workbook.active

        # Встановлення заголовків під час першого запуску
        if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
            sheet.cell(row=1, column=1, value="Название задачи")
            sheet.cell(row=1, column=2, value="Defect Density")
            sheet.cell(row=1, column=3, value="Customer Satisfaction (%)")

        # Запис метрик задач
        for i, task in enumerate(self.completed, start=2):
            sheet.cell(row=i, column=1, value=task.name)
            sheet.cell(row=i, column=2, value=task.defect_density)
            sheet.cell(row=i, column=3, value=task.customer_satisfaction)

        workbook.save(file_name)
        print(f"\nРезультати метрик збережено у файл {file_name}")

# Ініціалізація середовища SimPy
env = simpy.Environment()

# Створення проекту Kanban
project = KanbanProject(env)

# Додавання завдань до проекту
project.add_task("Розробка архітектури проєкту", 20)
project.add_task("Дизайн основного інтерфейсу", 32)
project.add_task("Розробка іконок та візуальних елементів", 24)
project.add_task("Дизайн графіків та діаграм", 24)
project.add_task("Дизайн повідомлень та повідомлень", 20)
project.add_task("Розробка головного меню та налаштувань", 40)
project.add_task("Розробка екрану додавання транзакцій", 40)
project.add_task("Створення функції додавання транзакцій", 40)
project.add_task("Категоризація транзакцій", 24)
project.add_task("Збереження даних користувача", 20)
project.add_task("Побудова графіків витрат", 40)
project.add_task("Автоматичне формування звітів", 24)
project.add_task("Система нагадувань про регулярні платежі", 24)
project.add_task("Інтеграція з банківськими API", 60)
project.add_task("Реалізація цілей заощаджень", 32)
project.add_task("Система захисту та шифрування даних", 20)
project.add_task("Налаштування багатофакторної автентифікації", 24)
```

```
project.add_task("Юніт-тестування основних функцій", 40)
project.add_task("Інтеграційне тестування", 40)
project.add_task("Стрес-тестування системи", 40)
project.add_task("Налагодження та оптимізація продуктивності", 40)
project.add_task("Інтеграція Google API для синхронізації даних", 16)
project.add_task("Інтеграція платіжних систем", 16)
project.add_task("Налаштування аналітичних інструментів", 32)
project.add_task("Впровадження системи PUSH-повідомлень", 24)
project.add_task("Додавання рейтингової системи користувачів", 32)

# Запуск симуляції
env.process(project.run_project())
env.run()

# Збереження результатів
project.save_time_to_excel()
project.save_metrics_to_excel()

# Виведення загального часу
print(f"\nКількість виконаних завдань: {len(project.completed)}")
print(f"\nЗагальний час, витрачений на проект Kanban: {project.total_time_spent} годин")
```

Waterfall_metric

```
import simpy
import random
from openpyxl import load_workbook, Workbook
import os


# Клас для завдань (Tasks)
class Task:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
        self.actual_duration = 0
        self.defect_density = 0
        self.customer_satisfaction = 0


# Клас для проекту Waterfall
class WaterfallProject:
    def __init__(self, env):
        self.env = env
        self.phases = []  # Список фаз проекту
        self.completed_phases = []  # Список завершених фаз
        self.total_time_spent = 0
```

```
    # Додаємо фазу проекту
    def add_phase(self, name, duration):
        self.phases.append(Task(name, duration))

    # Імітація випадкових відхилень у часі (±20%)
    def simulate_task_duration(self, base_duration):
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))  # Мінімум 1 година

    # Генерація метрик для фази
    def generate_phase_metrics(self, task):
        task.defect_density = round(random.uniform(0.1, 1.0), 2)  # Дефектність
        task.customer_satisfaction = round(random.uniform(60, 100), 2)  # Задоволеність

    # Запуск Waterfall проекту
    def run_project(self):
        for phase in self.phases:
            phase.actual_duration = self.simulate_task_duration(phase.duration)
            print(f"[{self.env.now}] Початок фази: {phase.name} (Фактичний час: {phase.actual_duration} годин)")
            yield self.env.timeout(phase.actual_duration)
            self.total_time_spent += phase.actual_duration


            self.generate_phase_metrics(phase)
            self.completed_phases.append(phase)


            print(f"[{self.env.now}] Фаза '{phase.name}' завершена. Defect Density: {phase.defect_density}, Customer Satisfaction: {phase.customer_satisfaction}%")

    # Запис часу виконання у файл Excel
    def save_time_to_excel(self, file_name="results_time_Waterfall.xlsx"):
        if os.path.exists(file_name):
            workbook = load_workbook(file_name)
        else:
            workbook = Workbook()
```

```
        workbook.active.title = "Time Results"

    sheet = workbook.active

    # Установлення заголовків при першому запуску
    if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
        sheet.cell(row=1, column=1, value="Назва фази")
        sheet.cell(row=1, column=2, value="Оцінка тривалості (год)")
        sheet.cell(row=1, column=3, value="Фактичний час (год)")

    # Записуємо дані фаз
    for i, phase in enumerate(self.completed_phases, start=2):
        sheet.cell(row=i, column=1, value=phase.name)
        sheet.cell(row=i, column=2, value=phase.duration)
        sheet.cell(row=i, column=3, value=phase.actual_duration)

    workbook.save(file_name)
    print(f"\nРезультати часу виконання збережено у файл {file_name}")

# Запис метрик у файл Excel
def save_metrics_to_excel(self, file_name="results_metrics_Waterfall.xlsx"):
    if os.path.exists(file_name):
        workbook = load_workbook(file_name)
    else:
        workbook = Workbook()
        workbook.active.title = "Metrics"

    sheet = workbook.active

    # Установлення заголовків при першому запуску
    if sheet.max_column == 1 and sheet.cell(row=1, column=1).value is None:
        sheet.cell(row=1, column=1, value="Назва фази")
        sheet.cell(row=1, column=2, value="Defect Density")
        sheet.cell(row=1, column=3, value="Customer Satisfaction (%)")

    # Запис метрик фаз
```

```
        for i, phase in enumerate(self.completed_phases, start=2):
            sheet.cell(row=i, column=1, value=phase.name)
            sheet.cell(row=i, column=2, value=phase.defect_density)
            sheet.cell(row=i, column=3, value=phase.customer_satisfaction)

        workbook.save(file_name)
        print(f"\nРезультати метрик збережено у файл {file_name}")

# Ініціалізація середовища SimPy
env = simpy.Environment()

# Створення проекту Waterfall
project = WaterfallProject(env)

# Додаємо повний список завдань до проекту
project.add_phase("Розробка архітектури проєкту", 20)
project.add_phase("Дизайн основного інтерфейсу", 32)
project.add_phase("Розробка іконок та візуальних елементів", 24)
project.add_phase("Дизайн графіків і діаграм", 24)
project.add_phase("Дизайн сповіщень та повідомлень", 20)
project.add_phase("Розробка головного меню та налаштувань", 40)
project.add_phase("Розробка екрану додавання транзакцій", 40)
project.add_phase("Створення функції додавання транзакцій", 40)
project.add_phase("Категоризація транзакцій", 24)
project.add_phase("Збереження даних користувача", 20)
project.add_phase("Побудова графіків витрат", 40)
project.add_phase("Автоматичне формування звітів", 24)
project.add_phase("Система нагадувань про регулярні платежі", 24)
project.add_phase("Інтеграція з банківськими API", 60)
project.add_phase("Реалізація цілей збережень", 32)
project.add_phase("Система захисту та шифрування даних", 20)
project.add_phase("Налаштування багатофакторної автентифікації", 24)
project.add_phase("Юніт-тестування основних функцій", 40)
project.add_phase("Інтеграційне тестування", 40)
project.add_phase("Стрес-тестування системи", 40)
project.add_phase("Налагодження та оптимізація продуктивності", 40)
```

```
project.add_phase("Інтеграція Google API для синхронізації даних", 16)
project.add_phase("Інтеграція платіжних систем", 16)
project.add_phase("Налаштування аналітичних інструментів", 32)
project.add_phase("Впровадження системи PUSH-повідомлень", 24)
project.add_phase("Додавання рейтингової системи користувачів", 32)

# Запускаємо симуляцію
env.process(project.run_project())
env.run()

# Запис результатів у два файли
project.save_time_to_excel()
project.save_metrics_to_excel()

# Виведення загального часу
print(f"\nКількість виконаних фаз: {len(project.completed_phases)}")
print(f"\nЗагальний час, витрачений на проект Waterfall: {project.total_time_spent} годин")
```

Scrum_size_team

```
import simpy
import random
import matplotlib.pyplot as plt

# Клас для завдань (Tasks)
class Task:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration

# Клас для проекту Scrum
class ScrumProject:
    def __init__(self, env, team_size, tasks_per_sprint):
        self.env = env
        self.team_size = team_size  # Розмір команди
        self.tasks_per_sprint = tasks_per_sprint
        self.backlog = []  # Завдання у беклозі
        self.completed_tasks = []  # Виконані завдання
        self.total_time_spent = 0
        self.communication_time_factor = self.calculate_communication_time_factor()  # Час на комунікацію залежить від розміру команди

    def calculate_communication_time_factor(self):
        # Час на комунікацію залежить від кількості пар учасників команди
        return 0.05 * (self.team_size * (self.team_size - 1))  # Коефіцієнт пропорційний кількості пар

    def add_task(self, name, duration):
        task = Task(name, duration)
        self.backlog.append(task)
```

```
    def simulate_task_duration(self, base_duration):
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))

    def run_sprint(self):
        tasks_completed = 0
        while self.backlog and tasks_completed < self.tasks_per_sprint:
            task = self.backlog.pop(0)
            actual_duration = self.simulate_task_duration(task.duration)
            actual_duration += self.communication_time_factor  # Додаємо час на комунікацію
            yield self.env.timeout(actual_duration)
            self.completed_tasks.append(task)
            self.total_time_spent += actual_duration
            tasks_completed += 1

    def run_project(self):
        sprint = 1
        while self.backlog:
            #print(f"\n=== Спринт {sprint} ===")
            yield self.env.process(self.run_sprint())
            #print(f"Завершено завдань: {len(self.completed_tasks)}")
            sprint += 1
        print(f"Загальний час виконання проєкту для команди з {self.team_size} учасників: {self.total_time_spent:.2f} годин")

# Ініціалізація середовища SimPy
def simulate_scrum(team_sizes, num_tasks, task_duration, task_types=None):
    results = []

    for team_size in team_sizes:
        env = simpy.Environment()
        project = ScrumProject(env, team_size, tasks_per_sprint=team_size * 2)  # Завдання на кожен спринт залежно від розміру команди

        # Генерація завдань з урахуванням типів
        for i in range(num_tasks):
            if task_types:
                task_type = random.choice(task_types)
                base_duration = task_duration * task_type['duration_multiplier']
                project.add_task(f"Task {i+1} ({task_type['name']})", base_duration)
            else:
                project.add_task(f"Task {i+1}", task_duration)

        env.process(project.run_project())
        env.run()

        results.append((team_size, len(project.completed_tasks), project.total_time_spent))

    return results

# Можливі типи завдань
TASK_TYPES = [
    {"name": "Проста", "duration_multiplier": 0.8},
    {"name": "Середня", "duration_multiplier": 1.0},
    {"name": "Складна", "duration_multiplier": 1.5}
```

```
]

# Виконання симуляції
team_sizes = [3, 5, 7, 10]  # Розміри команд
num_tasks = 30  # Загальна кількість завдань
task_duration = 8  # Середня тривалість одного завдання

results = simulate_scrum(team_sizes, num_tasks, task_duration, task_types=TASK_TYPES)

# Побудова графіків
team_sizes, completed_tasks, total_times = zip(*results)

# Графік виконаних завдань
plt.figure(figsize=(10, 5))
plt.bar(team_sizes, completed_tasks, color='skyblue')
plt.xlabel('Розмір команди')
plt.ylabel('Кількість виконаних завдань')
plt.title('Вплив розміру команди на кількість виконаних завдань (Scrum)')
plt.show()

# Графік загального часу
plt.figure(figsize=(10, 5))
plt.plot(team_sizes, total_times, marker='o', color='green')
plt.xlabel('Розмір команди')
plt.ylabel('Загальний час (години)')
plt.title('Вплив розміру команди на загальний час виконання проєкту (Scrum)')
plt.show()
```

Kanban_size_team

```
import simpy
import random
import matplotlib.pyplot as plt

# Клас для завдань (Tasks)
class Task:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration

# Клас для проекту Kanban
class KanbanProject:
    def __init__(self, env, team_size, wip_limit):
        self.env = env
        self.team_size = team_size  # Розмір команди
        self.wip_limit = wip_limit  # Ліміт задач у роботі
        self.backlog = []  # Завдання у беклозі
        self.in_progress = []  # Завдання в роботі
        self.completed_tasks = []  # Виконані завдання
        self.total_time_spent = 0
        self.communication_time_factor = self.calculate_communication_time_factor()  # Час на комунікацію

    def calculate_communication_time_factor(self):
        # Час на комунікацію залежить від кількості пар учасників команди
        return 0.05 * (self.team_size * (self.team_size - 1))

    def add_task(self, name, duration):
```

```
        task = Task(name, duration)
        self.backlog.append(task)

    def simulate_task_duration(self, base_duration):
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))

    def work_on_task(self, task):
        actual_duration = self.simulate_task_duration(task.duration)
        actual_duration += self.communication_time_factor  # Додаємо час на комунікацію
        yield self.env.timeout(actual_duration)
        self.completed_tasks.append(task)
        self.total_time_spent += actual_duration

    def run_project(self):
        while self.backlog or self.in_progress:
            # Переміщуємо завдання в роботу, якщо є вільне місце
            while len(self.in_progress) < self.wip_limit and self.backlog:
                task = self.backlog.pop(0)
                self.in_progress.append(task)

            # Виконуємо завдання в роботі
            if self.in_progress:
                task = self.in_progress.pop(0)
                yield self.env.process(self.work_on_task(task))

        print(f"Загальний час виконання проєкту для команди з {self.team_size} учасників: {self.total_time_spent:.2f} годин")

# Ініціалізація середовища SimPy
def simulate_kanban(team_sizes, num_tasks, task_duration, wip_limit, task_types=None):
    results = []

    for team_size in team_sizes:
        env = simpy.Environment()
        project = KanbanProject(env, team_size, wip_limit)

        # Генерація завдань із урахуванням типів
        for i in range(num_tasks):
            if task_types:
                task_type = random.choice(task_types)
                base_duration = task_duration * task_type['duration_multiplier']
                project.add_task(f"Task {i+1} ({task_type['name']})", base_duration)
            else:
                project.add_task(f"Task {i+1}", task_duration)

        env.process(project.run_project())
        env.run()

        results.append((team_size, len(project.completed_tasks), project.total_time_spent))

    return results

# Можливі типи завдань
TASK_TYPES = [
    {"name": "Проста", "duration_multiplier": 0.8},
```

```
    {"name": "Середня", "duration_multiplier": 1.0},
    {"name": "Складна", "duration_multiplier": 1.5}
]

# Виконання симуляції
team_sizes = [3, 5, 7, 10]  # Розміри команд
num_tasks = 30  # Загальна кількість завдань
task_duration = 8  # Середня тривалість одного завдання
wip_limit = 5  # Ліміт задач у роботі

results = simulate_kanban(team_sizes, num_tasks, task_duration, wip_limit, task_types=TASK_TYPES)

# Побудова графіків
team_sizes, completed_tasks, total_times = zip(*results)

# Графік виконаних завдань
plt.figure(figsize=(10, 5))
plt.bar(team_sizes, completed_tasks, color='skyblue')
plt.xlabel('Розмір команди')
plt.ylabel('Кількість виконаних завдань')
plt.title('Вплив розміру команди на кількість виконаних завдань (Kanban)')
plt.show()

# Графік загального часу
plt.figure(figsize=(10, 5))
plt.plot(team_sizes, total_times, marker='o', color='green')
plt.xlabel('Розмір команди')
plt.ylabel('Загальний час (години)')
plt.title('Вплив розміру команди на загальний час виконання проєкту (Kanban)')
plt.show()
```

Waterfall_size_team

```
import simpy
import random
import matplotlib.pyplot as plt

# Клас для фаз проєкту Waterfall
class Phase:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration

# Клас для проекту Waterfall
class WaterfallProject:
    def __init__(self, env, team_size):
        self.env = env
        self.team_size = team_size  # Розмір команди
        self.phases = []  # Фази проєкту
        self.completed_phases = []  # Завершені фази
        self.total_time_spent = 0
        self.communication_time_factor = self.calculate_communication_time_factor()  # Час на комунікацію

    def calculate_communication_time_factor(self):
        # Час на комунікацію залежить від кількості пар учасників команди
        return 0.05 * (self.team_size * (self.team_size - 1))  # Коефіцієнт пропорційний кількості пар
```

```
    def add_phase(self, name, duration):
        phase = Phase(name, duration)
        self.phases.append(phase)

    def simulate_phase_duration(self, base_duration):
        deviation = base_duration * 0.2
        actual_duration = base_duration + random.uniform(-deviation, deviation)
        return max(1, round(actual_duration))

    def run_project(self):
        for phase in self.phases:
            actual_duration = self.simulate_phase_duration(phase.duration)
            actual_duration += self.communication_time_factor  # Додаємо час на комунікацію
            #print(f"[{self.env.now}] Початок фази: {phase.name} (Фактичний час: {actual_duration:.2f} годин)")
            yield self.env.timeout(actual_duration)
            self.completed_phases.append(phase)
            self.total_time_spent += actual_duration
            #print(f"[{self.env.now}] Фаза '{phase.name}' завершена.")

        print(f"Загальний час виконання проєкту для команди з {self.team_size} учасників: {self.total_time_spent:.2f} годин")

# Ініціалізація середовища SimPy
def simulate_waterfall(team_sizes, phases):
    results = []

    for team_size in team_sizes:
        env = simpy.Environment()
        project = WaterfallProject(env, team_size)

        # Додавання фаз у проєкт
        for phase_name, phase_duration in phases:
            project.add_phase(phase_name, phase_duration)

        env.process(project.run_project())
        env.run()

        results.append((team_size, len(project.completed_phases), project.total_time_spent))

    return results

# Фази проєкту
PHASES = [
    ("Аналіз вимог", 40),
    ("Проєктування системи", 60),
    ("Розробка коду", 120),
    ("Тестування", 80),
    ("Впровадження", 40)
]

# Виконання симуляції
team_sizes = [3, 5, 7, 10]  # Розміри команд

results = simulate_waterfall(team_sizes, PHASES)

# Побудова графіків
```

```
team_sizes, completed_phases, total_times = zip(*results)

# Графік виконаних фаз
plt.figure(figsize=(10, 5))
plt.bar(team_sizes, completed_phases, color='skyblue')
plt.xlabel('Розмір команди')
plt.ylabel('Кількість виконаних фаз')
plt.title('Вплив розміру команди на кількість виконаних фаз (Waterfall)')
plt.show()

# Графік загального часу
plt.figure(figsize=(10, 5))
plt.plot(team_sizes, total_times, marker='o', color='green')
plt.xlabel('Розмір команди')
plt.ylabel('Загальний час (години)')
plt.title('Вплив розміру команди на загальний час виконання проєкту (Waterfall)')
plt.show()
```