

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів планування ІТ-проекту з розробки растрового графічного
редактора
(тема)

Виконав:

здобувач 2 року навчання,
групи УПГІТм-24-1

Нікіта КОНОПЛЯ
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проектами в
галузі інформаційних технологій

Керівник доцент каф. ІУС Тетяна БІЛОВА
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ІУС


(підпис)

Костянтин ПЕТРОВ
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Інформаційних управляючих систем _____

Рівень вищої освіти _____ другий (магістерський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)Тип програми _____ освітньо-наукова _____
(освітньо-професійна або освітньо-наукова)Освітня програма Управління проектами в галузі інформаційних технологій
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

« 06 » квітня 20 26 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУздобувачеві _____ Коноплі Никиті Сергійовичу _____
(прізвище, ім'я, по батькові)

- Тема роботи Дослідження методів планування ІТ-проєкту з розробки растрового графічного редактора
затверджена наказом університету від 23 березня 2026 р. № 231Ст
- Термін подання здобувачем роботи до екзаменаційної комісії 21 травня 2026 р.
- Вихідні дані до роботи науково-технічні публікації, монографії та інтернет-ресурси з тематики планування ІТ-проєктів; матеріали щодо розробки растрових графічних редакторів, матеріали науково-дослідної практики.
- Перелік питань, що потрібно опрацювати в роботі Аналіз існуючих методів управління ІТ-проєктами з розробки растрових графічних редакторів та виявлення їх обмежень для інді-сегмента. Розробка теоретичних основ удосконаленого гібридного методу планування. Формальний опис методу, його ролей, артефактів і процедур. Системний аналіз ризиків та обмежень застосування методу. Формування системи метрик ефективності методу. Розробка інформаційної технології підтримки методу із шаблонами документів та чек-листами валідації. Вибір та конфігурація програмних засобів імплементації методу. Експериментальна перевірка методу шляхом порівняльного моделювання з альтернативними методами управління.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної області та існуючих методів управління ІТ-проектами	01.04.2026– 06.04.2026	виконано
2	Аналіз ринку растрових графічних редакторів та визначення вимог до метод	06.04.2026– 13.04.2026	виконано
3	Розробка теоретичних основ та формальний опис методу DTP-Indie	13.04.2026– 20.04.2026	виконано
4	Формування системи метрик ефективності методу	20.04.2026– 29.04.2026	виконано
5	Розробка інформаційної технології підтримки методу та BPMN-діаграми	29.04.2026– 04.05.2026	виконано
6	Конфігурація програмних засобів реалізації методу (Notion, Trello)	04.05.2026– 11.05.2026	виконано
7	Програмна реалізація методу та його експериментальна перевірка	11.05.2026– 13.05.2026	виконано
8	Формулювання висновків та оформлення пояснювальної записки	13.05.2026– 15.05.2026	виконано
9	Рецензування роботи	15.05.2026– 18.05.2026	виконано
10	Передзахист	19.05.2026– 20.05.26	виконано
11	Подання роботи до екзаменаційної комісії	20.05.2026– 21.05.26	виконано

Дата видачі завдання 06 квітня 2026 р.

Здобувач _____

(підпис)

Керівник роботи _____

(підпис)

доцент Тетяна БІЛОВА

(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка містить 138 сторінок, 20 рисунків, 6 таблиць, 9 формул, 17 джерел.

ГІБРИДНИЙ МЕТОД, ГНУЧКІ МЕТОДОЛОГІЇ, ГРАФІЧНИЙ РЕДАКТОР, ДВОКОЛІЄВЕ ПЛАНУВАННЯ, ІНДІ-РОЗРОБКА, ІТ-ПРОЄКТ, ПЛАНУВАННЯ.

Об'єктом дослідження кваліфікаційної роботи є процеси управління ІТ-проєктами в контексті створення програмних продуктів невеликими інді-командами.

Метою дослідження є розробка удосконаленого гібридного методу планування для ефективного створення растрового графічного редактора невеликими інді-командами.

Методи дослідження – метод просторової декомпозиції для розділення архітектурної та функціональної складових проєкту, методи предиктивного та адаптивного управління, а також експеримент і моделювання для практичної перевірки інформаційної технології.

В ході виконання роботи розроблено гібридний метод планування DTP-Indie, що базується на просторовій декомпозиції проєкту на дві паралельні колії – Колію Ядра з предиктивним управлінням архітектурою та Колію Інструментарію з адаптивним управлінням розробкою користувацького функціоналу. Також створена інформаційна технологія підтримки процесу планування на основі інструментів Notion і Trello, та проведена експериментальна перевірка.

Наукова новизна роботи полягає в розробці методу DTP-Indie, який, на відміну від існуючих підходів, включає трирівневий гейт UX-валідації з кількісними метриками юзабіліті та формалізовану процедуру запитів на корекцію ядра як механізм керованого технічного боргу.

ABSTRACT

The explanatory note contains 138 pages, 20 figures, 6 tables, 9 equations, 17 sources.

AGILE METHODOLOGIES, DUAL-TIER PLANNING, GRAPHICS EDITOR, HYBRID METHOD, INDIE DEVELOPMENT, IT PROJECT, PLANNING.

The object of research of this thesis is IT project management processes in the context of software product creation by small indie teams.

The aim of the study is to develop an improved hybrid planning method for the efficient creation of a raster graphics editor by small indie teams.

Research methods include the spatial decomposition method for separating the architectural and functional components of the project, predictive and adaptive management methods, as well as experimentation and modeling for the practical validation of the information technology.

During the course of the study, the DTP-Indie hybrid planning method was developed. It is based on the spatial decomposition of the project into two parallel tracks: the Core Track, with predictive architecture management, and the Tooling Track, with adaptive management of user functionality development. Additionally, an information technology to support the planning process was created using Notion and Trello, and experimental validation was conducted.

The scientific novelty of the study lies in the development of the DTP-Indie method, which, unlike existing approaches, incorporates a three-level UX validation gate with quantitative usability metrics and a formalized procedure for core correction requests as a mechanism for managed technical debt.

ЗМІСТ

Скорочення та умовні позначки.....	9
Вступ.....	11
1 Аналіз процесу планування ІТ-проєктів з розробки графічних редакторів.....	13
1.1 Аналіз властивостей об'єкта дослідження.....	13
1.1.1 Структура процесу як об'єкта управління	13
1.1.2 Класифікація проєктів з розробки растрових графічних редакторів	14
1.1.3 Управлінські проблеми в процесах розробки растрових редакторів	18
1.2 Аналіз класичних методів управління ІТ-проєктами	20
1.2.1 Предиктивні методи управління	21
1.2.2 Гнучкі методології методи управління (Agile).....	22
1.2.3 Ощадливі підходи до розробки	24
1.2.4 Узагальнення та порівняльна характеристика.....	25
1.3 Аналіз гібридних методів планування	27
1.3.1 Метод Water-Scrum-Fall	28
1.3.2 Бімодальна модель управління	29
1.3.3 Disciplined Agile Delivery.....	30
1.3.4 Scaled Agile Framework	31
1.3.5 Узагальнена характеристика гібридних методів	32
1.4 Постановка задачі дослідження.....	34
2 Теоретичні основи удосконаленого гібридного методу планування	38
2.1 Підхід до вирішення задачі планування розробки графічного редактора..	38
2.1.1 Загальний принцип просторової декомпозиції замість темпоральної.....	38
2.1.2 Критерій декомпозиції: зв'язність компонентів	40
2.1.3 Принцип формалізованого зворотного зв'язку між коліями	41
2.1.4 Принцип ресурсної ефективності для інді-команд.....	43
2.1.5 Загальний алгоритм методу DTP-Indie	44
2.2 Удосконалений гібридний метод планування DTP-Indie	46

2.2.1 Базова архітектура методу	47
2.2.2 Колія Ядра: предиктивне управління архітектурними компонентами	48
2.2.3 Колія Інструментарію	49
2.2.4 Гейт UX-валідації: трирівневий контроль якості досвіду	50
2.2.5 Запит на корекцію ядра і Гейт перевірки технічного боргу	53
2.2.6 Ролі, артефакти та церемонії методу	55
2.2.7 Завершальна фаза і життєвий цикл після релізу	56
2.2.8 Сфера застосування методу	57
2.3 Управління ризиками та обмеження методу DTP-Indie	59
2.3.1 Класифікація ризиків методу	59
2.3.2 Структурні обмеження методу	63
2.3.3 Узагальнення: рекомендації щодо застосування методу	65
2.4 Система метрик ефективності методу DTP-Indie	66
2.4.1 Структура системи метрик	67
2.4.2 Метрики процесу управління	68
2.4.3 Метрики якості продукту	70
2.4.4 Інтегральний показник ефективності методу	73
3 Інформаційна технологія підтримки методу DTP-Indie	76
3.1 Інформаційна технологія підтримки методу	76
3.1.1 Загальна структура інформаційної технології	76
3.1.2 Формалізований опис процесу у нотації BPMN	77
3.1.3 Шаблон документа бачення продукту	79
3.1.4 Шаблон плану розробки ядра	79
3.1.5 Шаблон беклогу інструментарію та журналу запитів на корекцію	80
3.1.6 Чек-листи гейтів валідації	80
3.1.7 Типові робочі процедури	82
3.1.8 Інтеграція компонентів інформаційної технології	84
3.2 Інструментарій імплементації інформаційної технології	85
3.2.2 Конфігурація робочого простору Notion	88
3.2.3 Конфігурація дошки Trello	97

3.2.4 Інтеграція Notion і Trello через крос-посилання.....	100
3.2.5 Типовий робочий тиждень менеджера за методом DTP-Indie.....	101
3.2.6 Альтернативні стеки інструментарію для масштабованих команд.....	103
4 Програмна реалізація та експериментальна перевірка	105
4.1 Програмна реалізація методу DTP-Indie.....	105
4.1.1 Підхід до програмної реалізації.....	105
4.1.2 Управлінська платформа на основі Notion і Trello.....	106
4.1.3 BPMN 2.0 як описова мова процесу	107
4.1.4 Інші технологічні рішення	108
4.2 Експериментальна перевірка методу DTP-Indie	109
4.2.1 Методика експериментальної перевірки.....	109
4.2.2 Експертна оцінка характеристик методів	110
4.2.3 Інтерпретація показників порівняння	111
4.2.4 Обмеження методу та сфера його доцільного застосування	113
4.2.5 Узагальнення результатів експериментальної перевірки	114
Висновки	116
Перелік джерел посилання	119
Додаток А Графічний матеріал кваліфікаційної роботи.....	121
Додаток Б Шаблон. Документ бачення продукту.....	134
Додаток В Шаблон. План розробки ядра	136
Додаток Г Шаблон. Беклог інструментарію	137
Додаток Ґ Шаблон. Журнал запитів на корекцію ядра	138

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Колія Інструментарію – паралельний трек методу DTP-Indie, орієнтований на адаптивну розробку інструментів користувацького функціоналу.

Колія Ядра – паралельний трек методу DTP-Indie, орієнтований на предиктивну розробку архітектурно стабільного графічного движка.

ACR (Architectural Change Request) – формалізований запит на корекцію ядра, що подається за необхідності зміни архітектури в методі DTP-Indie.

API (Application Programming Interface) – інтерфейс прикладного програмування, який дозволяє різним програмам взаємодіяти та безпечно обмінюватися даними між собою.

BPMN (Business Process Model and Notation) – міжнародний стандарт графічної нотації, який використовується для наочного моделювання та опису бізнес-процесів за допомогою стандартизованих символів, що дозволяє легко зрозуміти логіку роботи як людям, так і технічним системам.

DAD (Disciplined Agile Delivery) – людино-орієнтований та орієнтований на навчання гібридний підхід, який об'єднує найкращі практики Scrum, Kanban та інших гнучких методологій.

DTP-Indie (Dual-Tier Planning for Indie) – розроблений у дослідженні гібридний двоколієвий метод планування IT-проектів для інді-команд.

EA (Estimation Accuracy) – метрика точності планування, що відображає відхилення між оціненим та фактично витраченим часом на виконання завдань.

FIR (Flow Interruption Rate) – частота переривання робочого процесу, яка вимірює кількість вимушених зупинок або збоїв користувача під час виконання сценарію.

FPS (Frames Per Second) – показник частоти оновлення кадрів на екрані за одну секунду, що визначає плавність відображення графічного інтерфейсу.

Lean (Lean Software Development) – ошадлива розробка; методологія створення програмного забезпечення, головна мета якої — усунення будь-яких втрат задля максимального підвищення цінності продукту для клієнта.

RCR (Rework Cycle Rate) – показник частоти доопрацювань, який визначає частку завдань, що повертаються на повторну розробку чи виправлення помилок.

SLP (Stroke Latency Perception) – показник сприйняття затримки введення, який визначає суб'єктивну межу помітності запізнення візуального відгуку на дію користувача.

SAFe (Scaled Agile Framework) – це система практик для масштабування гнучких методів розробки на рівні великих корпорацій.

SUS (System Usability Scale) – метрика оцінювання сприйняття користувачами зручності та простоти використання інтерфейсу за допомогою опитувальника.

TDI (Technical Debt Index) – індекс технічного боргу, що кількісно оцінює обсяг і зусилля, необхідні для рефакторингу та усунення дефектів у коді.

TFS (Time-to-First-Stroke) – метрика швидкості відгуку інтерфейсу, яка вимірює час від дії користувача до першої візуальної зміни на екрані.

UI (User Interface) – користувацький інтерфейс; зовнішній вигляд, оформлення та фізичне втілення програмного забезпечення, з яким взаємодіє людина.

UX (User Experience) – користувацький досвід; рівень зручності та інтуїтивності роботи з програмним забезпеченням.

XP (eXtreme Programming) – екстремальне програмування; гнучка методологія розробки програмного забезпечення, спрямована на швидке створення високоякісного коду в умовах постійних змін.

ВСТУП

У сучасних умовах цифрової трансформації економіки стрімко зростає попит на програмні продукти, орієнтовані на обробку мультимедійного контенту та забезпечення творчої діяльності користувачів. Растровий графічний редактор як об'єкт управління є складною інформаційною системою з багаторівневою архітектурою, що включає математичні алгоритми обробки зображень, систему управління пам'яттю, механізми роботи з шарами та безпосередньо інтерфейс користувача. Розробка подібного програмного продукту характеризується підвищеною складністю з управлінської точки зору, оскільки вимагає поєднання високої продуктивності системи зі зручністю та інтуїтивністю користувацького досвіду.

Еволюція провідних графічних редакторів демонструє перехід від жорсткого планування функціоналу до ітеративних принципів розробки, що дозволяє швидко адаптувати продукти під потреби ринку. Водночас на ринку формується стійкий запит на спеціалізовані інструменти цифрової графіки з мінімальним порогом входження, де базові функції знаходяться на поверхні і не потребують від користувача професійного навчання. Створення такого інтуїтивного програмного забезпечення вимагає нових управлінських підходів, оскільки традиційні методи не завжди здатні забезпечити баланс між стабільністю складного ядра та гнучкістю розробки інтерфейсу.

Традиційні підходи до планування ІТ-проєктів не забезпечують достатній рівень керованості при створенні систем, у яких вимоги до інтерфейсу формуються поступово, а архітектурні рішення (наприклад, графічний рушій) мають довгострокові наслідки. Каскадна модель забезпечує формалізацію вимог та архітектурну цілісність, але її жорсткість обмежує можливість тестування гіпотез щодо користувацького досвіду. Гнучкі методології дозволяють скорочувати ризики невідповідності очікуванням аудиторії, проте їх

застосування без стратегічного фундаменту при розробці графічного програмного забезпечення призводить до фрагментарності рішень та накопичення технічного боргу. Це зумовлює актуальність формування адаптованої гібридної моделі управління, орієнтованої на специфіку розробки растрового графічного редактора в умовах інді-команди.

Метою роботи є вдосконалення методу планування ІТ-проєкту з розробки растрового графічного редактора шляхом створення спеціалізованої гібридної двоколієвої моделі управління, що забезпечує баланс між архітектурною стабільністю та інтуїтивністю користувацького досвіду.

Наукова новизна отриманих результатів полягає у формуванні спеціалізованого методу планування ІТ-проєкту, адаптованого до особливостей створення растрових графічних редакторів інді-командами. На відміну від універсальних методів управління, запропонований підхід передбачає просторову декомпозицію проєкту на дві паралельні колії з різними парадигмами управління – предиктивну для архітектурного ядра та адаптивну для розробки користувацького інструментарію, з формалізованим механізмом зворотного зв'язку через запити на корекцію ядра, що дозволяє ефективно управляти технічним боргом та забезпечити низький поріг входження для кінцевого користувача.

Практична значущість отриманих результатів полягає у можливості застосування розробленого методу при управлінні стартап-проєктами у сфері креативного програмного забезпечення, а також у діяльності невеликих (інді) команд розробників розміром 2–7 осіб, що працюють над створенням інструментів цифрової графіки, для оптимізації термінів розробки та зниження організаційних ризиків.

Роботу представлено й апробовано на 30-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» [1]

1 АНАЛІЗ ПРОЦЕСУ ПЛАНУВАННЯ ІТ-ПРОЄКТІВ З РОЗРОБКИ ГРАФІЧНИХ РЕДАКТОРІВ

1.1 Аналіз властивостей об'єкта дослідження

Об'єктом дослідження виступає процес планування ІТ-проєкту з розробки растрового графічного редактора як специфічний клас управлінських задач у сфері створення креативного програмного забезпечення. Для коректного формулювання задачі удосконалення методу планування необхідно проаналізувати структуру цього процесу, виявити його ключові властивості та класифікувати типові управлінські проблеми, що виникають у проєктах даного класу.

1.1.1 Структура процесу як об'єкта управління

Процес планування розробки растрового графічного редактора має чітко виражену двокомпонентну структуру, що відрізняє його від інших класів ІТ-проєктів. У формальному вигляді цей процес можна описати схемою «вхід – перетворення – вихід – обмеження».

Вхідними даними процесу планування виступають функціональні вимоги до редактора (перелік необхідних інструментів, фільтрів, форматів файлів), нефункціональні вимоги (продуктивність обробки растрових матриць, споживання пам'яті, стабільність роботи), ергономічні вимоги (поріг входження для нового користувача, інтуїтивність інтерфейсу) та ресурсні обмеження команди розробки.

Послідовність дій (перетворення) включає аналіз вимог, вибір технологічного стеку, проєктування архітектури графічного ядра, планування

інструментарію користувача, координацію розробки ядра та інтерфейсу, контроль якості та валідацію.

Вихідними даними процесу є план проекту з розподілом задач у часі, архітектурна специфікація ядра, беклог задач з розробки інтерфейсу та система метрик контролю прогресу.

Обмеженнями процесу виступають критична залежність якості продукту від ергономіки інтерфейсу, неможливість зміни базових архітектурних рішень на пізніх етапах без значного рефакторингу та потреба у безперервному тестуванні UX на реальних користувачах.

Ключовою особливістю об'єкта дослідження є наявність двох якісно різних компонентів системи з принципово різними вимогами до управління. Перший компонент, графічне ядро (Engine), характеризується високою зв'язністю (high coupling) між модулями: математична модель растрової матриці, система виділення пам'яті, алгоритми рендерингу та логіка шарів утворюють жорстко пов'язану структуру. Другий компонент – інструментарій користувача (Toolset), навпаки, складається з відносно незалежних елементів (пензлі, фільтри, інструменти виділення) з низькою зв'язністю, але високою чутливістю до користувацького досвіду.

Ця структурна асиметрія об'єкта дослідження є визначальною: різні компоненти потребують різних управлінських парадигм, що формує методологічне підґрунтя для гібридного підходу до планування.

1.1.2 Класифікація проєктів з розробки растрових графічних редакторів

Для подальшого аналізу необхідно класифікувати проєкти даної предметної області за критеріями, які впливають на вибір методу планування. Емпіричною базою для класифікації слугує еволюція відомих продуктів ринку,

кожен з яких репрезентує характерну управлінську модель свого історичного періоду та сегмента.

Розгляд еволюції управлінських підходів логічно розпочати з Microsoft Paint – продукту, що тривалий час залишався еталоном базового растрового редактора. Його розробка велася в умовах, коли життєвий цикл графічного застосунку був жорстко синхронізований з релізами операційної системи, що обумовлювало природний вибір каскадної моделі управління. Така синхронізація гарантувала стабільність та інтеграцію з платформою, проте водночас породжувала характерну патологію – архітектурну стагнацію, за якої функціональні можливості продукту еволюціонували значно повільніше за потреби користувачів. Цей кейс демонструє граничний випадок передбачуваності управління, що досягається ціною втрати конкурентоспроможності.

Принципово іншу управлінську парадигму репрезентує Adobe Photoshop, який упродовж кількох десятиліть утримує статус індустріального стандарту професійного растрового редагування. Розробка цього продукту відбувається в умовах корпоративного масштабу зі значною командою інженерів, що уможливило застосування комплексних гібридних практик: стратегічне планування архітектурних змін здійснюється за предиктивним підходом, тоді як розробка нових інструментів та функцій ведеться ітеративно. Управлінська ефективність такої моделі підтверджується ринковим домінуванням продукту, проте її зворотним боком стало накопичення феномену feature bloat – функціональної надлишковості, що системно підвищує поріг входження для нових користувачів та робить інтерфейс громіздким. Кейс Photoshop важливий для дослідження як ілюстрація того, що навіть формалізований гібридний підхід без вбудованого механізму контролю ергономічної доцільності призводить до деградації UX (Досвід користувача) у довгостроковій перспективі.

Відмінний управлінський архетип демонструє проєкт GIMP – найвідоміший приклад open-source растрового редактора, розробка якого

ведеться децентралізованою спільнотою без єдиного власника продукту. Відсутність централізованого стратегічного управління інтерфейсом призвела до того, що історично GIMP розвивався за принципом «дизайну, орієнтованого на розробника», коли архітектурні та інтерфейсні рішення приймалися виходячи з технічної логіки, а не з потреб користувача. Як наслідок, продукт здобув репутацію функціонально потужного, але неінтуїтивного інструменту з високим порогом входження. Цей кейс має критичне значення для обґрунтування необхідності UX-валідації у методі планування: відсутність формалізованого контролю ергономіки на рівні процесу управління закономірно призводить до її деградації незалежно від кваліфікації окремих учасників команди.

Особливий інтерес становить проєкт Krita як приклад еволюції управлінської моделі open-source-продукту. На ранньому етапі (до середини 2000-х) розробка велася за традиційним для open-source неформальним підходом, що породжувало проблеми, аналогічні кейсу GIMP. Проте у подальшому проєкт пройшов суттєву трансформацію: було створено Krita Foundation як інституційну основу, запроваджено регулярне краудфандингове фінансування, а процес розробки структуровано через єдину систему управління задачами та регулярні робочі зустрічі ключових учасників. Цей перехід ілюструє важливу для дослідження тезу: навіть проєкти, що історично розвивались у логіці open-source, у міру зростання вимог до якості продукту змушені імплементувати елементи структурованого управління. Кейс Krita підтверджує, що повна відмова від формалізованих управлінських практик несумісна зі створенням конкурентоспроможного графічного редактора.

Найрелевантнішим для дослідження є кейс Procreate – растрового редактора, створеного невеликою інді-командою без зовнішнього інвестування. Цей продукт демонструє, що інді-команди здатні створювати графічні редактори, конкурентні за UX з продуктами корпоративного масштабу. Водночас задокументовані обставини розробки – багатократні переписування кодової бази, численні цикли перегляду дизайну, тривалий період до релізу –

свідчать про те, що успіх досягався переважно завдяки інтуїції засновників, а не завдяки застосуванню формалізованого методу планування. Більше того, керівництво студії публічно характеризує власний підхід як такий, що базується на інтуїтивних рішеннях та свідомо протистоїть каскадній моделі. З погляду проектного менеджменту, такий підхід має фундаментальну вразливість: він не є відтворюваним. Іншими словами, успіх Procreate не може бути систематично повторений іншими інді-командами, оскільки інтуїція засновників не піддається передачі через методологічні артефакти. Саме цей розрив між емпіричною ефективністю інді-розробки та відсутністю її методологічного оформлення формує практичну потребу, на задоволення якої спрямоване дослідження. Систематизована порівняльна характеристика проаналізованих проєктів наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика проєктів з розробки растрових графічних редакторів

Продукт	Розмір команди	Модель управління	Цільовий сегмент	Ключова управлінська проблема
Microsoft Paint	Корпоративна	Каскадна модель	Базові користувачі ОС	Архітектурна стагнація
Adobe Photoshop	Корпоративна (велика)	Гібридна (Water-Scrum-Fall)	Професійний ринок	Feature bloat, високий поріг входження
GIMP	Розподілена спільнота	Неформальний open-source	Технічно підготовлені користувачі	Криза юзабіліті, відсутність UX-стратегії
Krita	Гібридна (Foundation + спільнота)	Структурований open-source з Гнучкі методології	Цифрові художники	Залежність від донорського фінансування
Procreate	Інді (мала)	Інтуїтивний ітеративний підхід	Цифрові художники на планшетах	Відсутність формалізованого методу

Аналіз таблиці 1.1 виявляє характерну закономірність: жоден з проєктів, що належить до сегмента малих команд (Krita на ранньому етапі, Procreate), не використовує формалізованого гібридного методу планування. Ці продукти досягли успіху значною мірою завдяки інтуїції засновників, що, однак, не гарантує відтворюваності результату для нових проєктів даного класу.

1.1.3 Управлінські проблеми в процесах розробки растрових редакторів

На основі проведеного аналізу можна сформулювати ключові управлінські проблеми, що виникають у процесах розробки растрових графічних редакторів і потребують вирішення на рівні методу планування.

Проблема 1. Конфлікт між архітектурною жорсткістю ядра та гнучкістю розробки інтерфейсу. Графічне ядро редактора (растрова матриця, система шарів, рендерер) має властивість високої зв'язності модулів. Зміна базових структур даних на пізніх етапах призводить до каскадного рефакторингу, що еквівалентно повторній розробці. Водночас інтерфейс редактора потребує безперервного тестування на реальних користувачах та ітеративного доопрацювання, що принципово несумісне з лінійним плануванням. Як наслідок, виникає управлінська дилема: жорстке планування блокує розвиток ергономіки, а гнучке планування ставить під загрозу стабільність ядра.

Проблема 2. Відсутність формалізованих критеріїв валідації UX перед інтеграцією. У всіх проаналізованих кейсах рішення про прийняття або відхилення нового елемента інтерфейсу приймалось на основі суб'єктивного судження керівника продукту або консенсусу команди. Відсутність формальних критеріїв (порогових значень метрик юзабіліті, чек-листів ергономічної оцінки) призводить до двох типів помилок: впровадження інтерфейсних рішень, що

погіршують UX, або відмова від технічно якісних рішень через суб'єктивні упередження.

Проблема 3. Невідповідність існуючих гібридних методів специфіці інді-команд. Корпоративні гібридні підходи розраховані на команди десятків і сотень розробників з виділеним менеджментом UI/UX (Користувацький інтерфейс), окремою архітектурною командою та налагодженими процесами комунікації. Інді-команди, у яких один розробник часто поєднує ролі архітектора, програміста та дизайнера інтерфейсу, не можуть застосовувати ці методи без значної адаптації. Емпіричним підтвердженням є кейс Procreate, де команда свідомо відмовилась від каскадної моделі без формалізованої альтернативи.

Проблема 4. Ризик накопичення технічного боргу при швидких ітераціях UI/UX. Коли інтерфейсні зміни вимагають модифікації ядра (наприклад, новий тип пензля потребує нової операції змішування на рівні рендерера), команда стикається з вибором: або витратити час на правильну інтеграцію через рефакторинг ядра, або реалізувати компромісне рішення для дотримання термінів спринту. Без формалізованого механізму контролю архітектурної цілісності інді-команди систематично обирають другий варіант, що призводить до накопичення технічного боргу та зниження стабільності продукту в довгостроковій перспективі.

Виявлені управлінські проблеми мають спільну природу: вони виникають через відсутність методу планування, який би одночасно враховував специфіку об'єкта (двокомпонентна структура з різними вимогами до компонентів) та обмеження суб'єкта управління (інді-команда з обмеженими ресурсами). Класичні методології управління IT-проєктами та існуючі гібридні підходи, аналіз яких проводиться у наступних підрозділах, не вирішують цю задачу в повному обсязі.

Об'єкт дослідження – процес планування IT-проєкту з розробки растрового графічного редактора – характеризується двокомпонентною структурою з якісно різними вимогами до управління графічним ядром та інструментарієм

користувача. Аналіз практичних кейсів (Microsoft Paint, Photoshop, GIMP, Krita, Procreate) виявив чотири ключові управлінські проблеми: конфлікт між архітектурною жорсткістю та гнучкістю UI, відсутність формалізованої UX-валідації, невідповідність корпоративних гібридних методів інді-командам, ризик накопичення технічного боргу при ітеративній розробці. Виявлені проблеми визначають практичні задачі, які повинен вирішувати удосконалений метод планування для проєктів даного класу.

1.2 Аналіз класичних методів управління ІТ-проєктами

Для формування ефективного підходу до планування проєкту з розробки растрового графічного редактора необхідно проаналізувати існуючі універсальні методології управління ІТ-проєктами з точки зору їх застосовності до специфіки об'єкта дослідження. Як було показано у попередньому підрозділі, цей об'єкт характеризується двокомпонентною структурою (графічне ядро та інструментарій користувача) з якісно різними вимогами до управління, що суттєво обмежує застосовність будь-якого однорідного методу. Аналіз класичних методів проведено у логіці їх історичного формування – від предиктивних моделей до адаптивних та гнучких філософій – з акцентом на виявлення сильних сторін та принципових обмежень кожного підходу у контексті розробки графічного програмного забезпечення малими командами.

1.2.1 Предиктивні методи управління

Предиктивні методи об'єднує спільна управлінська філософія: вимоги до продукту фіксуються на ранніх етапах життєвого циклу, після чого процес розробки слідує заздалегідь визначеному плану з мінімальними відхиленнями. Ця парадигма склалася історично як перенесення інженерних практик будівництва та машинобудування у сферу розробки програмного забезпечення і протягом десятиліть домінувала в управлінні великими ІТ-проєктами.

Каскадна модель є найвідомішим представником цієї парадигми. Вона передбачає лінійне послідовне виконання етапів життєвого циклу проєкту: збір вимог, проєктування, кодування, тестування, впровадження та підтримка, причому перехід до наступного етапу відбувається лише після повного завершення попереднього. Управлінська цінність каскадної моделі для розробки графічного редактора виявляється передусім на рівні архітектурного ядра. Проєктування математичних моделей растрової матриці, системи виділення пам'яті та алгоритмів рендерингу – це задачі з високою вартістю помилки, оскільки невдалий вибір базових структур даних на ранньому етапі тягне за собою каскадний рефакторинг усього продукту. Жорстка фіксація архітектурних рішень за каскадною моделлю мінімізує цей ризик. Водночас принципове обмеження методу полягає у тому, що тестування продукту проводиться лише наприкінці циклу, що унеможлиблює своєчасний зворотний зв'язок щодо ергономіки інтерфейсу. Для растрового редактора, конкурентоспроможність якого критично залежить від інтуїтивності UI, цей недолік є фатальним: продукт, що технічно працює коректно, ризикує виявитись непридатним до використання.

V-подібна модель є розширенням каскадного підходу, у якому кожному етапу проєктування ставиться у відповідність паралельний етап верифікації. Метод історично формувався як галузевий стандарт для систем з критичними вимогами до безпеки – медичного обладнання, авіоніки, систем управління

транспорт [2]. Для розробки растрового графічного редактора V-подібна модель є надмірно бюрократизованою: вимоги до тотального документування кожного кроку та формалізованої верифікації абсолютно непропорційні характеру задачі і фактично блокують будь-яку гнучкість у роботі з UI/UX. У контексті інді-розробки даний метод не є практично придатним.

Спіральна модель, запропонована Б. Боемом, ускладнює лінійну логіку каскаду введенням обов'язкового аналізу ризиків та прототипування на кожному витку розробки. Це робить її теоретично привабливою для проєктування інноваційних графічних рушіїв, де технологічна невизначеність є суттєвим фактором. Проте практична імплементація методу передбачає значні витрати на формалізовану оцінку ризиків та прототипування, що робить його економічно недоцільним для команд з обмеженими ресурсами. Спіральна модель залишається інструментом великих корпоративних проєктів і не може бути ефективно застосована у форматі інді-розробки.

Узагальнюючи аналіз предиктивних методів, можна констатувати, що вони забезпечують високий рівень керованості та архітектурної цілісності, проте принципово несумісні з вимогами до гнучкості розробки користувацького інтерфейсу. Для проєкту з розробки растрового графічного редактора це означає, що предиктивні методи можуть бути ефективно застосовані лише до управління розробкою графічного ядра, але не до проєкту в цілому.

1.2.2 Гнучкі методології методи управління (Agile)

Філософія гнучких методологій (Agile) сформувалась у відповідь на виявлені обмеження предиктивних підходів і будується на принципово іншій парадигмі: визнанні неможливості повної формалізації вимог на ранніх етапах та орієнтації на ітеративну адаптацію продукту до реальних потреб користувача.

Замість єдиного послідовного плану розробка ведеться короткими циклами з регулярним переглядом пріоритетів, що теоретично відповідає характеру задачі проектування інтерфейсу растрового редактора.

Scrum є найпоширенішим фреймворком гнучкої розробки, що структурує процес у вигляді коротких ітерацій (спринтів) тривалістю від одного до чотирьох тижнів. Кожен спринт завершується працездатним інкрементом продукту, який може бути продемонстрований замовнику або кінцевому користувачу для отримання зворотного зв'язку [3]. Для розробки інтерфейсу растрового редактора Scrum виявляється надзвичайно ефективним: він дозволяє створити прототип конкретного інструмента (наприклад, нового типу пензля або фільтра), миттєво протестувати його на реальних користувачах та скоригувати ергономіку до інтеграції у фінальний продукт. Водночас застосування чистого Scrum до проекту в цілому несе критичний ризик. Оскільки фреймворк не передбачає формалізованого етапу архітектурного планування, а пріоритезація беклогу зорієнтована на коротко- та середньострокову цінність для користувача, базові архітектурні рішення приймаються «по дорозі», у міру виникнення відповідних задач. Для системи з високою зв'язністю ядра, якою є графічний редактор, це закономірно призводить до накопичення технічного боргу та архітектурної фрагментації, описаних у підрозділі 1.1.3 як проблема 4.

Kanban репрезентує іншу гілку Agile-філософії – підхід безперервного потоку, у якому замість фіксованих за часом ітерацій основним механізмом контролю виступає обмеження кількості задач, що знаходяться у роботі одночасно. Цей метод є високоефективним на етапі підтримки існуючого продукту, виправлення помилок та поступового додавання дрібних поліпшень. Для початкової розробки растрового редактора Kanban має суттєве обмеження: він не містить вбудованих механізмів масштабного планування створення архітектури з нуля та не задає стратегічного бачення продукту. У контексті дослідження Kanban може розглядатись як корисний допоміжний інструмент для пізніх стадій життєвого циклу, але не як самостійний метод планування.

Екстремальне програмування (eXtreme Programming, XP) є найжорсткішим за інженерною дисципліною гнучкою методологією. Він фокусується на конкретних практиках написання коду: розробці через тестування Test-Driven Development, постійному рефакторингу, парному програмуванні, безперервній інтеграції. У контексті розробки растрового редактора Test-Driven Development має очевидну цінність для забезпечення коректності математичних алгоритмів змішування кольорів, накладання шарів та обчислення фільтрів. Проте обов'язкова практика парного програмування є економічно недоцільною для невеликих команд: вона фактично подвоює людино-години на одиницю функціоналу, що неприйнятно за умов обмежених ресурсів інді-розробки. Загальна вимога до високої інженерної дисципліни робить XP виснажливим для малих команд у довгостроковій перспективі.

Аналіз адаптивних методів виявляє управлінський патерн, дзеркальний до предиктивних підходів. Гнучкі методології забезпечують ідеальні умови для розробки користувацького інтерфейсу через швидкі цикли зворотного зв'язку, проте їх застосування до управління архітектурою графічного ядра створює критичні ризики стабільності продукту. Іншими словами, адаптивні методи добре працюють там, де погано працюють предиктивні, і навпаки.

1.2.3 Ощадливі підходи до розробки

Ощадлива розробка (Lean Software Development) являє собою адаптацію виробничих принципів Toyota Production System до сфери розробки програмного забезпечення. На відміну від Scrum або каскадної моделі, Lean є передусім управлінською філософією, а не структурованим алгоритмом дій: основними принципами виступають усунення втрат (відмова від зайвого функціоналу та проміжних артефактів, що не створюють цінності для користувача), посилення

навчання команди, відкладення рішень до моменту, коли вони можуть бути прийняті на основі максимально повної інформації, та максимізація цінності для кінцевого користувача [4].

Філософська спорідненість Lean із концепцією растрового редактора з мінімальним порогом входження є очевидною: обидва підходи орієнтовані на боротьбу з функціональним перевантаженням і фокус на справді необхідних можливостях.

Водночас з погляду методу планування Lean має принципове обмеження – він задає образ мислення команди та критерії оцінки рішень, але не пропонує покрокового алгоритму створення архітектури графічного ядра з нуля. Lean може ефективно доповнювати інший метод як ціннісний орієнтир, але не може виконувати функцію самостійного методу планування ІТ-проєкту.

1.2.4 Узагальнення та порівняльна характеристика

Систематизована порівняльна характеристика проаналізованих класичних методів за критеріями, релевантними для розробки растрового графічного редактора інді-командою, наведена у таблиці 1.2.

Аналіз таблиці 1.2 виявляє фундаментальну закономірність: жоден з класичних однорідних методів не задовольняє вимоги до розробки растрового графічного редактора у повному обсязі. Предиктивні методи забезпечують стабільність архітектури ціною ергономічної стагнації; адаптивні методи гарантують якісний UX ціною загрози архітектурної фрагментації; ощадлива розробка задає правильні ціннісні орієнтири, але не пропонує структурованого алгоритму планування. Виявлене розмежування сильних сторін методів за компонентами об'єкта дослідження (ядро та інтерфейс) формує природне

обґрунтування для гібридного підходу, який поєднував би предиктивні принципи у стратегічному управлінні архітектурою з адаптивними практиками у розробці інтерфейсу. Аналіз існуючих гібридних методів та їх застосовності до інді-розробки проводиться у наступному підрозділі.

Класичні методи управління ІТ-проектами поділяються на дві принципово різні парадигми – предиктивну та адаптивну, кожна з яких ефективно вирішує лише частину задачі управління розробкою растрового графічного редактора.

Таблиця 1.2 – Порівняльна характеристика класичних методів управління

Метод	Підходить для розробки ядра	Підходить для розробки UI/UX	Сильні сторони	Принципові обмеження
Каскадна модель	Так	Ні	Архітектурна цілісність, мінімізація вартості помилки	Відсутність зворотного зв'язку щодо UX до фінального етапу
V-подібна модель	Частково	Ні	Суворі верифікації на кожному етапі	Надмірна бюрократизація
Спіральна модель	Так	Частково	Систематичне управління ризиками	Економічна недоцільність для малих команд
Scrum	Ні	Так	Швидкий зворотний зв'язок, адаптація до вимог користувача	Накопичення технічного боргу при роботі з системами високої зв'язності
Kanban	Ні	Частково	Гнучкість, контроль обсягу робіт у потоці	Відсутність механізмів стратегічного планування
XP	Частково	Так	Забезпечує коректність математичних алгоритмів	Парне програмування подвоює витрати ресурсів
Lean	Як філософія	Як філософія	Боротьба з роздуттям функцій, фокус на цінності	Не є самостійним методом планування

Предиктивні методи забезпечують архітектурну цілісність графічного ядра, але блокують гнучкість розробки інтерфейсу. Адаптивні методи створюють умови для якісного UX через швидкий зворотний зв'язок, але несуть критичний ризик архітектурної фрагментації при роботі з системами високої зв'язності. Ощадлива розробка задає важливі ціннісні принципи, але не функціонує як самостійний метод планування [5].

Виявлена комплементарність сильних та слабких сторін предиктивних і адаптивних підходів обґрунтовує необхідність переходу до гібридних методів управління, орієнтованих на специфіку об'єкта дослідження.

1.3 Аналіз гібридних методів планування

Усвідомлення обмеженості як предиктивних, так і адаптивних методів управління, продемонстроване у попередньому підрозділі, призвело до появи на ринку проєктного менеджменту цілої сім'ї гібридних підходів. Об'єднуючим принципом усіх гібридних методів є спроба поєднати стабільність формалізованого планування з гнучкістю ітеративної розробки шляхом одночасного або послідовного застосування елементів різних парадигм у межах одного проєкту. Для дослідження ці методи становлять особливий інтерес, оскільки потенційно можуть забезпечити необхідне розмежування управління архітектурним ядром та інтерфейсом растрового графічного редактора. Однак ключовим критерієм оцінки гібридних методів у даному дослідженні є їх застосовність до умов інді-розробки – невеликої команди з обмеженими ресурсами, де відсутні спеціалізовані ролі менеджерів, архітекторів та UX-дизайнерів.

1.3.1 Метод Water-Scrum-Fall

Water-Scrum-Fall є де-факто найпоширенішим гібридним підходом у сучасному корпоративному ІТ-секторі та водночас найочевиднішим відображенням ідеї поєднання предиктивних і адаптивних практик [6]. Сутність методу полягає у послідовному застосуванні трьох управлінських парадигм на різних фазах життєвого циклу проекту: каскадної моделі – на етапах ініціації, збору вимог та проєктування архітектури; фреймворку Scrum – безпосередньо для основної фази написання коду; знову каскадної моделі – для координованого релізу та передачі продукту в експлуатацію [7].

Управлінська привабливість Water-Scrum-Fall у контексті розробки растрового графічного редактора є очевидною на концептуальному рівні. Початкова каскадна фаза дозволяє ретельно спроектувати архітектуру графічного ядра, що відповідає вимозі стабільності системи високої зв'язності. Середня Scrum-фаза забезпечує гнучкість розробки інтерфейсу та інструментарію користувача. Завершальна каскадна фаза гарантує контрольований реліз продукту [8]. На перший погляд, метод виглядає природним рішенням для задач, виявлених у підрозділі 1.1.3.

Проте детальний аналіз методу виявляє його принципову непридатність для умов інді-розробки. Перше обмеження стосується ресурсних вимог: обидві каскадні фази передбачають значне формалізоване документування – технічні специфікації архітектури, плани релізу, документацію впровадження, – що фактично потребує наявності виділених ролей системного архітектора та технічного писаря. Інді-команди такими ресурсами не володіють. Друге обмеження є концептуальним: Water-Scrum-Fall є послідовним методом, у якому фази йдуть строго одна за одною. Якщо під час Scrum-фази виявиться, що ергономічні вимоги до інтерфейсу конфліктують з прийнятими архітектурними рішеннями, метод не передбачає формалізованого механізму повернення до

архітектурної фази без зриву всього плану. Це особливо критично для розробки растрового редактора, де ергономічні відкриття часто виникають саме в процесі прототипування і потребують зміни базових архітектурних рішень. Третє обмеження стосується довжини циклу: Water-Scrum-Fall історично формувався для проєктів з довгими реліз-циклами, де між початковим архітектурним плануванням і фінальним релізом проходять місяці або роки [9]. Інді-команди, навпаки, виграють від частих публічних релізів, що несумісно з тривалою фінальною каскадною фазою методу.

1.3.2 Бімодальна модель управління

Концепція бімодальних ІТ, запропонована аналітичною компанією Gartner, базується на принципово іншому управлінському рішенні: замість послідовного чергування парадигм у часі вона передбачає їх паралельне співіснування у межах однієї організації. Команда розробки розділяється на два функціональні режими: Режим 1 фокусується на стабільності, надійності та передбачуваності та керується за предиктивною моделлю; Режим 2 орієнтований на інновації, швидку адаптацію та експерименти і керується за адаптивними принципами [10].

Теоретична відповідність бімодальної моделі задачам розробки растрового графічного редактора є практично ідеальною. Команда Режиму 1 могла б займатися розробкою графічного ядра – рендерера, системи шарів, базових структур даних – за каскадними принципами з повним документуванням і верифікацією. Команда Режиму 2, паралельно і незалежно, вела б розробку інструментарію користувача за Scrum-методологією з частими ітераціями та користувацьким тестуванням. Основна проблема Water-Scrum-Fall –

необхідність послідовного переключення між парадигмами – у бімодальній моделі знімається повністю.

Однак саме у цій теоретичній перевазі і криється практична непридатність методу для інді-розробки. Бімодальна модель за визначенням потребує фактичного організаційного розділення команди на дві незалежні підгрупи, кожна з яких має власне керівництво та власний стиль управління. У корпоративному середовищі з командами у десятки людей таке розділення є природним. В умовах інді-команди з двох-трьох розробників, де часто один спеціаліст відповідає одночасно за математику обробки кольору та за поведінку інтерфейсних кнопок, бімодальна модель є фізично не імплементованою. Більше того, навіть якщо команда розширюється до п'яти-семи осіб, штучне розділення на два режими породжує комунікаційні бар'єри між людьми, які раніше працювали в єдиному просторі рішень. Бімодальна модель залишається інструментом великих організацій і не може бути коректно масштабована донизу до інді-сегмента.

1.3.3 Disciplined Agile Delivery

Disciplined Agile Delivery (DAD) представляє найбільш зрілий і концептуально опрацьований гібридний підхід серед розглянутих. Метод був розроблений Скоттом Емблером і Марком Лайнсом наприкінці 2000-х років, отримав широке поширення після публікації методологічного посібника у 2012 році і у 2019 році був придбаний Інститутом Управління Проєктами, що остаточно закріпило його статус як офіційно підтриманого PMI-стандарту гібридного управління.

Принципова відмінність DAD від Water-Scrum-Fall і бімодальної моделі полягає у відмові від жорсткого структурного розділення фаз або команд на

користь концепції «способу роботи», що адаптується до контексту конкретного проекту [11]. Метод інтегрує практики Scrum, Lean, Kanban, XP та Agile-модельовання у єдиній рамці, охоплюючи увесь життєвий цикл – від ініціації через побудову до переходу в експлуатацію. Замість фіксованого алгоритму DAD пропонує орієнтовану на цілі структуру, у якій команда сама обирає конкретні практики під свій контекст.

Для задачі розробки растрового графічного редактора DAD теоретично надає високу гнучкість: фаза ініціації дозволяє провести стратегічне планування архітектури, фаза побудови може бути організована як комбінація Scrum-спринтів для UI та більш формалізованих практик для ядра, фаза експлуатації забезпечує контрольований реліз. Проте практичне застосування DAD до інді-розробки наштовхується на його фундаментальну характеристику – орієнтацію на корпоративний сегмент. Адаптація DAD до інді-команди вимагає значних зусиль з відсіювання непридатних практик і реконструкції методу під обмежений контекст. Іншими словами, DAD є набором будівельних блоків, з якого теоретично можна зібрати потрібний метод, але сама ця збірка є нетривіальною методологічною задачею, що виходить за межі компетенцій типової інді-команди.

1.3.4 Scaled Agile Framework

Scaled Agile Framework (SAFe) є найрозвиненішим методом масштабування Agile-практик на рівень великих організацій з кількома командами розробки. SAFe пропонує багаторівневу структуру управління – від рівня окремої команди через рівень програми і портфеля до рівня великого підприємства, причому кожен рівень має власні артефакти, церемонії та ролі.

У контексті дослідження SAFe згадується для повноти аналізу гібридних методів, проте його застосовність до інді-розробки не потребує детального обґрунтування. Метод проектувався під ситуації, коли над одним продуктом працюють десятки команд по кілька десятків людей кожна [12]. Рівень формалізації, кількість ролей та обсяг церемоній SAFe не лише непропорційні масштабу інді-розробки – вони фундаментально несумісні з її організаційною природою. SAFe ілюструє граничну точку розвитку гібридних методів у бік корпоративного масштабування і опосередковано підтверджує тезу про методологічний розрив між корпоративним та інді-сегментами.

1.3.5 Узагальнена характеристика гібридних методів

Систематизована характеристика проаналізованих гібридних методів за критеріями застосовності до інді-розробки растрового графічного редактора наведена у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика гібридних методів планування у контексті інді-розробки

Метод	Цільовий масштаб	Принцип поєднання парадигм	Сильні сторони	Обмеження для інді-розробки
Water-Scrum-Fall	Великі та середні корпоративні проекти	Послідовне у часі	Простота розуміння, поширеність у галузі	Бюрократія каскадних фаз, відсутність зворотних зв'язків
Бімодальна модель	Великі організації	Паралельне у командах	Незалежність потоків	Поділ команди на дві групи

Кінець таблиці 1.3

Метод	Цільовий масштаб	Принцип поєднання парадигм	Сильні сторони	Обмеження для інді-розробки
DAD	Enterprise-масштаб	Контекстно-залежний	Гнучкість, охоплення повного життєвого циклу	Складність адаптації до малого контексту
SAFe	Дуже великі організації	Багаторівневе масштабування Agile	Координація багатьох команд	Принципова невідповідність масштабу інді-розробки

Аналіз таблиці виявляє системну закономірність, що формує ключову проблему дослідження. Усі чотири проаналізовані гібридні методи історично формувалися у контексті корпоративної розробки і несуть відбиток своєї генези: вони передбачають значні людські ресурси, формалізовані ролі, розгалужену документацію та тривалі цикли релізу. Спроби застосувати ці методи до інді-розробки растрового графічного редактора призводять або до критичного перевантаження команди адміністративними витратами (Water-Scrum-Fall, DAD), або до фізичної неможливості імплементації через обмежений склад команди (Бімодальна модель, SAFe). Виявлений розрив між корпоративним і інді-сегментами проєктного менеджменту не є технічним непорозумінням, що може бути усунене простим спрощенням існуючих методів. Він має методологічну природу і відображає фундаментальну відсутність наукової роботи з адаптації гібридних принципів до умов малих команд, що працюють над специфічними класами продуктів – зокрема, растровим графічним програмним забезпеченням.

Існуючі гібридні методи планування IT-проєктів (Water-Scrum-Fall, Бімодальна модель, DAD, SAFe) демонструють методологічну зрілість і концептуальну привабливість для розробки систем з різномірними компонентами, до яких належить і растровий графічний редактор. Однак усі ці

методи спроектовані для корпоративного контексту і несумісні з умовами індивідуальної розробки за двома ключовими критеріями: ресурсна вимогливість (формалізовані ролі, документація, церемонії) та організаційна структура (розділення команди на спеціалізовані групи). Виявлений методологічний розрив обґрунтовує практичну необхідність розробки удосконаленого ресурсоефективного гібридного методу планування, орієнтованого на специфіку розробки растрових графічних редакторів індивідуальними командами. Постановка відповідної задачі дослідження здійснюється у наступному підрозділі.

1.4 Постановка задачі дослідження

Проведений у попередніх підрозділах аналіз об'єкта дослідження, класичних методів управління ІТ-проектами та існуючих гібридних підходів дозволяє чітко ідентифікувати фундаментальну проблему планування у сфері розробки растрового графічного програмного забезпечення малими командами та сформулювати науково-практичну задачу її вирішення.

Об'єктом дослідження є процес планування ІТ-проекту з розробки растрового графічного редактора як специфічний клас управлінських задач, що характеризується двокомпонентною структурою – необхідністю одночасного управління архітектурно стабільним графічним ядром та ергономічно адаптивним інструментарієм користувача.

Предметом дослідження є методи планування ІТ-проектів, які забезпечують поєднання предиктивних і адаптивних управлінських парадигм у контексті розробки растрового графічного редактора невеликими командами розробників (індивідуальний сегмент).

Проблема дослідження полягає у протиріччі між практичними задачами розробки та можливостями існуючих методологій управління.

З одного боку, сучасний ринок креативного програмного забезпечення формує стійкий запит на спеціалізовані растрові графічні редактори з мінімальним порогом входження для кінцевого користувача. Розробка такого продукту вимагає одночасного вирішення двох якісно різних інженерно-управлінських задач: створення стабільного оптимізованого ядра обробки растрових даних, що потребує жорсткого планування та архітектурної дисципліни, та проєктування інтуїтивного користувацького інтерфейсу, що потребує безперервного тестування на реальних користувачах та ітеративного доопрацювання [13].

З іншого боку, проведений аналіз методів управління ІТ-проєктами демонструє їх системну неспроможність ефективно вирішити цю подвійну задачу в умовах обмежених ресурсів інді-команд. Класичні предиктивні методи (каскадна модель, v-подібна модель, спіральна модель) забезпечують стабільність архітектури, але блокують гнучкість роботи з ергономікою інтерфейсу. Адаптивні методології (Scrum, Kanban, XP) створюють умови для якісного UX через швидкий зворотний зв'язок, але несуть критичний ризик архітектурної фрагментації при роботі з системами високої зв'язності. Існуючі гібридні підходи (Water-Scrum-Fall, Бімодальна модель, DAD, SAFe) концептуально вирішують проблему ізоляції процесів управління різними компонентами, проте усі вони історично формувалися для корпоративного контексту і несуть відбиток своєї генези – значні ресурсні вимоги, формалізовані ролі, розгалужену документацію, що робить їх практичну імплементацію в інді-сегменті фізично та економічно недоцільною.

Кейс-аналіз практичних прикладів розробки растрових графічних редакторів (Microsoft Paint, Adobe Photoshop, GIMP, Krita, Procreate) додатково підтвердив виявлений методологічний розрив: успішні інді-проєкти у даному жанрі досягали результату переважно завдяки інтуїції засновників, а не завдяки застосуванню формалізованого методу планування, що ставить під сумнів відтворюваність такого успіху для нових команд .

Постановка задачі дослідження. Виходячи з виявленого протиріччя, формулюється науково-практична задача: розробити удосконалений ресурсоефективний гібридний метод планування ІТ-проєкту, який забезпечує баланс між стабільністю архітектури графічного ядра та гнучкістю розробки користувацького інтерфейсу при створенні растрового графічного редактора в умовах обмежених ресурсів інді-команди, з вбудованими формалізованими механізмами валідації ергономічної якості та контролю архітектурної цілісності.

Відповідно до поставленої задачі, подальше дослідження спрямоване на послідовне вирішення таких підзадач:

1. Розробка теоретичного підходу до вирішення задачі планування ІТ-проєкту з розробки растрового графічного редактора інді-командою на основі принципу декомпозиції управлінських процесів за критерієм зв'язності компонентів системи (підрозділ 2.1).

2. Формування удосконаленого гібридного методу планування з двоконтурною валідацією, що поєднує предиктивні принципи управління графічним ядром та адаптивні принципи управління інструментарієм користувача із формалізованими механізмами UX-валідації та архітектурного контролю (підрозділ 2.2).

3. Проєктування інформаційної технології підтримки розробленого методу, що забезпечує його практичну імплементацію в робочих процесах інді-команди (підрозділ 3.1).

4. Розробка інструментарію імплементації інформаційної технології з урахуванням ресурсних обмежень та технологічного стеку, доступного для інді-команд (підрозділ 3.2).

5. Програмна реалізація компонентів технології підтримки розробленого методу планування (підрозділ 4.1).

6. Експериментальна перевірка ефективності розробленого методу шляхом порівняння з класичними підходами управління за визначеною системою кількісних метрик (підрозділ 4.2).

Послідовне вирішення сформульованих підзадач становить логічну структуру подальшого дослідження і забезпечує перехід від теоретичного обґрунтування методу до його практичної верифікації.

Сформульовано задачу для дослідження, що полягає у розробці удосконаленого ресурсоефективного гібридного методу планування ІТ-проєкту з розробки растрового графічного редактора інді-командою. Визначено об'єкт і предмет дослідження, обґрунтовано наукову проблему через виявлене протиріччя між практичними задачами розробки та можливостями існуючих методологій. Сформульовано шість підзадач дослідження, що відповідають структурі подальших розділів роботи.

Сформульовано задачу для дослідження – розробити удосконалений ресурсоефективний гібридний метод планування ІТ-проєкту з розробки растрового графічного редактора інді-командою із вбудованими механізмами валідації ергономічної якості та контролю архітектурної цілісності. Визначено шість підзадач, що становлять структуру подальших розділів роботи.

2 ТЕОРЕТИЧНІ ОСНОВИ УДОСКОНАЛЕНОГО ГІБРИДНОГО МЕТОДУ ПЛАНУВАННЯ

2.1 Підхід до вирішення задачі планування розробки графічного редактора

Сформульована у підрозділі 1.4 науково-практична задача потребує розробки удосконаленого гібридного методу планування, який забезпечує одночасне виконання двох суперечливих вимог: збереження архітектурної цілісності графічного ядра та збереження ергономічної гнучкості користувацького інтерфейсу. Перш ніж переходити до формального опису методу у наступному підрозділі, необхідно обґрунтувати загальний теоретичний підхід до вирішення задачі та сформулювати ключові принципи, на яких базуватиметься метод DTP-Indie (Dual-Tier Planning for Indie – дворівневе планування для інді-команд).

2.1.1 Загальний принцип просторової декомпозиції замість темпоральної

Аналіз існуючих методів управління, проведений у розділі 1, виявив принциповий недолік усіх однорідних підходів: вони застосовують єдину управлінську парадигму до цілого проєкту, ігноруючи неоднорідну структуру об'єкта розробки. Гібридні методи усувають цей недолік частково – вони визнають необхідність поєднання різних парадигм, але реалізують це поєднання через темпоральну (часову) декомпозицію проєкту на послідовні фази.

Найвідомішим прикладом темпоральної декомпозиції є модель Water-Scrum-Fall, описана в аналітичному звіті Forrester у 2011 році [14]. У цій моделі проєкт послідовно проходить три фази: початкову Water-фазу (планування і архітектура за каскадною моделлю), середню Scrum-фазу (розробка за гнучкою

методологією) та фінальну Fall-фазу (реліз і підтримка за каскадною моделлю). Принципово важливим є той факт, що Forrester не розглядає Water-Scrum-Fall як рекомендований метод – у звіті прямо констатується, що це опис патологічного стану, у якому опинилися більшість корпоративних організацій через інерцію бюрократичних процесів планування і релізу. Водночас, незважаючи на критичне ставлення авторів до цієї моделі, вона де-факто залишається найпоширенішою формою гібридного управління у корпоративному ІТ-секторі і часто помилково сприймається як легітимний метод.

Принциповий недолік темпоральної декомпозиції полягає у тому, що вона зберігає однорідність парадигми всередині кожної фази. Під час Scrum-фази Water-Scrum-Fall усі компоненти системи розробляються за єдиними правилами, незалежно від їх внутрішньої природи. Для проєктів з однорідною архітектурою це прийнятно. Для проєктів з принципово неоднорідною структурою – таких, як растровий графічний редактор – це призводить до системних проблем, описаних у підрозділі 1.1.3.

В основу запропонованого підходу DTP-Indie покладено принципово іншу ідею – просторова декомпозиція замість темпоральної. Замість того щоб ділити проєкт на послідовні у часі фази, проєкт ділиться на дві паралельні підсистеми, кожна з яких має власну управлінську парадигму, що зберігається протягом усього життєвого циклу. Перша підсистема (Колія Ядра) керується предиктивними принципами, друга (Колія Інструментарію) – адаптивними. Обидві підсистеми працюють одночасно і взаємодіють через формалізований механізм синхронізації.

Ключова відмінність від Water-Scrum-Fall полягає у тому, що в DTP-Indie в будь-який момент часу проєкт перебуває не в одній фазі (Water, Scrum або Fall), а у двох паралельних режимах одночасно. Питання «у якій фазі знаходиться проєкт» некоректне – натомість релевантним є питання «у якому стані знаходиться кожна з колій». Це усуває фундаментальне обмеження темпоральних гібридів – неможливість одночасно мати архітектурну

стабільність і UX-гнучкість, оскільки в темпоральній моделі ці властивості належать різним фазам.

2.1.2 Критерій декомпозиції: зв'язність компонентів

Запропонована просторова декомпозиція потребує об'єктивного критерію розділення системи на підсистеми. Довільне розділення (наприклад, за функціональними модулями або за організаційними ролями) не забезпечує методологічної обґрунтованості. Тому в основу декомпозиції покладено фундаментальну характеристику програмних систем – зв'язність компонентів, концепція якої була введена в програмну інженерію ще у 1974 році Л. Константіном і В. Стівенсом.

Зв'язність визначає міру взаємозалежності між модулями системи: висока зв'язність означає, що зміна в одному модулі тягне за собою зміни у багатьох інших, низька зв'язність означає, що модулі є відносно незалежними і можуть змінюватися окремо. Для растрового графічного редактора рівень зв'язності компонентів суттєво відрізняється у різних частинах системи, що і формує природну межу для декомпозиції.

Компоненти з високою зв'язністю формують графічне ядро редактора. До них належать: математична модель растрової матриці, система виділення пам'яті для роботи з зображеннями, базові алгоритми рендерингу, система шарів та їх композиції, механізми undo/redo, кольорові простори. Зміна базових структур даних у будь-якому з цих компонентів призводить до каскадного оновлення усіх залежних модулів. Це означає, що рішення про архітектуру цих компонентів повинні прийматися ретельно і з мінімальною кількістю переробок.

Компоненти з низькою зв'язністю формують інструментарій користувача. До них належать: окремі інструменти малювання (пензлі, олівці, гумки),

інструменти виділення, графічні фільтри, експортні модулі, елементи інтерфейсу. Кожен з цих компонентів спирається на Application Programming Interface (API) ядра, але є відносно незалежним від інших компонентів свого рівня. Доопрацювання, заміна або видалення окремого інструмента не впливає на стабільність решти системи. Це означає, що для цих компонентів допустимий – і навіть бажаний – ітеративний характер розробки з регулярним перевизначенням пріоритетів.

Виявлене розшарування системи за рівнем зв'язності формує методологічно обґрунтовану основу для просторової декомпозиції. Колія Ядра охоплює усі компоненти з високою зв'язністю і керується предиктивно. Колія Інструментарію охоплює усі компоненти з низькою зв'язністю і керується адаптивно. На відміну від довільного організаційного поділу команди (як у Бімодальній моделі від Gartner), запропонована декомпозиція спирається на об'єктивну властивість самої системи і не залежить від конкретного складу команди розробки.

2.1.3 Принцип формалізованого зворотного зв'язку між коліями

Просторова декомпозиція з двома паралельними коліями створює нову управлінську проблему, відсутню в темпоральних гібридах: необхідність синхронізації роботи колій між собою. У реальному проєкті розробки растрового редактора часто виникають ситуації, коли робота над компонентом інструментарію виявляє необхідність зміни компонента ядра. Найтиповіший приклад: новий інструмент малювання потребує операції змішування кольорів, яку поточна архітектура рендерера не підтримує. У такій ситуації Колія Інструментарію не може просто продовжувати роботу – необхідне втручання у Колію Ядра, плановане відповідно до її предиктивної логіки.

У темпоральних гібридних методах механізм такого втручання принципово відсутній. Як прямо констатує Forrester у звіті 2011 року, інформаційний потік у Water-Scrum-Fall є однонапрямленим: вимоги, прийняті у Water-фазі, передаються у Scrum-фазу для реалізації, але зворотного формального каналу для перегляду цих вимог не передбачено [15]. Аналогічна ситуація у бімодальній моделі управління, де Команда Режиму 1 і Команда Режиму 2 працюють організаційно ізольовано, що також блокує формалізований обмін інформацією. DAD декларує контекстну адаптивність, але не пропонує конкретного механізму крос-канальної синхронізації для випадку, коли паралельні потоки розробки потребують взаємного впливу.

Третім принципом методу DTP-Indie є запровадження двостороннього формалізованого каналу зв'язку між коліями. Цей канал реалізується через дві контрольні точки (далі – гейти), що працюють у протилежних напрямках: гейт UX-валідації перевіряє інструменти на виході Колії Інструментарію, а гейт перевірки технічного боргу контролює запити на корекцію ядра, що надходять з Колії Інструментарію у Колію Ядра.

Ключовою інновацією методу є можливість запиту на корекцію ядра – формалізованої процедури, через яку Колія Інструментарію може ініціювати зміну в Engine Roadmap (плані розробки ядра). Якщо в процесі UX-валідації нового інструмента виявляється, що його коректна реалізація потребує модифікації архітектури ядра, формується запит на корекцію. Цей запит проходить процедуру перевірки технічного боргу і, у разі схвалення, призводить до офіційного перепланування Engine Roadmap. Тимчасово робота Колії Ядра переключається на реалізацію корекції, після чого повертається до основного плану. Колія Інструментарію у цей час продовжує роботу над інструментами, що не залежать від запиту на корекцію.

Запит на корекцію ядра не є аналогом неформального «давайте перепланує», характерного для Scrum-фази у Water-Scrum-Fall. Це формалізована процедура з документованими критеріями прийняття рішення,

журналом усіх запитів і чіткими правилами впливу на план. Саме формалізація цього зворотного зв'язку перетворює DTP-Indie з простого паралельного гібриду на повноцінний метод управління з керованим механізмом адаптації.

2.1.4 Принцип ресурсної ефективності для інді-команд

Окремим методологічним викликом, відсутнім у класичних та корпоративних гібридних методах, є необхідність забезпечити роботоспроможність методу в умовах малих команд. Як показав аналіз у підрозділі 1.3, існуючі гібридні підходи передбачають значні ресурсні витрати на адміністрування процесу: виокремлені ролі менеджера, архітектора, дизайнера, технічного писаря; розгалужену документацію; численні церемонії узгодження. Інді-команди, що складаються з двох-семи розробників, не мають можливості виділити людський ресурс на повноцінне обслуговування таких процесів.

Принцип ресурсної ефективності DTP-Indie реалізується через свідому мінімізацію адміністративного навантаження на команду. Кількість обов'язкових ролей зведена до трьох (Власник методу, Розробник ядра, Розробник інструментарію), причому ці ролі є функціональними, а не організаційними – одна особа може поєднувати дві або всі три функції залежно від розміру команди. Документація обмежена трьома базовими артефактами: документом бачення продукту, планом розробки ядра та беклогом інструментарію з журналом запитів на корекцію. Кількість регулярних церемоній зведена до двох: щотижнева синхронізація колій та валідаційна сесія наприкінці спринту інструментарію.

Принципово важливим є те, що ресурсна ефективність DTP-Indie досягається не через спрощення Water-Scrum-Fall або інших корпоративних методів – таке спрощення неминуче призводило б до втрати ключових

механізмів контролю якості. Натомість ресурсна ефективність вбудована у саму архітектуру методу: трирівневий гейт UX-валідації заміщує собою цілий відділ якості, формалізований запит на корекцію заміщує комітет архітектурного управління, паралельна організація колій усуває необхідність у складній координації між фазами.

2.1.5 Загальний алгоритм методу DTP-Indie

Розглянуті принципи інтегруються в узагальнений алгоритм роботи методу, який охоплює повний життєвий цикл проєкту з розробки растрового графічного редактора. Алгоритм складається з трьох фаз: підготовчої, основної (паралельної) та завершальної.

1) Підготовча фаза. Команда формує три базові артефакти: документ бачення продукту, що визначає цільову аудиторію, ключові функціональні вимоги та обмеження; план розробки ядра, що визначає послідовність етапів створення графічного движка з оцінками тривалості; беклог інструментарію, що містить пріоритезований перелік інструментів, фільтрів та функцій інтерфейсу. На цьому ж етапі визначаються цільові порогові значення UX-метрик для майбутньої валідації інструментів.

2) Основна (паралельна) фаза. Колія Ядра і Колія Інструментарію працюють одночасно. Розробник ядра послідовно реалізує етапи плану розробки ядра у предиктивному режимі без переключень. Розробник інструментарію працює спринтами по 1–2 тижні, обираючи з беклогу ті інструменти, що підтримуються поточним станом ядра. Кожен інструмент після завершення проходить трирівневий гейт UX-валідації. У разі необхідності змін у ядрі формується запит на корекцію, який проходить гейт перевірки технічного боргу.

Схвалені запити призводять до офіційного перепланування плану ядра. Раз на тиждень проводиться синхронізаційна зустріч колій.

3) Завершальна фаза. Після того як основна частина функціоналу розроблена і пройшла валідацію, проєкт переходить до завершальної фази, що включає підготовку релізу, інтеграційне тестування, бета-тестування на реальних користувачах та публічний випуск продукту. На цьому етапі обидві колії переходять у режим стабілізації: Колія Ядра фокусується на оптимізації продуктивності та виправленні критичних дефектів, Колія Інструментарію – на доопрацюванні інтерфейсу за результатами зворотного зв'язку від бета-тестерів. Після релізу проєкт переходить у режим підтримки, у якому метод DTP-Indie продовжує застосовуватися для випуску оновлень: великі архітектурні оновлення проходять через план розробки ядра, нові інструменти та поліпшення інтерфейсу – через беклог інструментарію з обов'язковою UX-валідацією.

2.1.6 Сумісність розробленого підходу з вимогами інді-сегмента

Запропонований підхід до вирішення задачі планування забезпечує одночасне виконання трьох ключових вимог, виявлених у попередніх підрозділах і не задоволених жодним з існуючих методів [16]. По-перше, він гарантує архітектурну стабільність графічного ядра через предиктивне управління Колії Ядра. По-друге, він забезпечує ергономічну гнучкість користувацького інтерфейсу через адаптивне управління Колії Інструментарію та формалізовану UX-валідацію. По-третє, він залишається придатним для застосування малими командами завдяки свідомій мінімізації адміністративного навантаження.

Підхід також зберігає здатність до масштабування. У межах інді-сегменту (2–7 осіб) метод працює через поєднання функціональних ролей. У разі росту

команди до 8–15 осіб ролі можуть розподілятися між окремими спеціалістами без зміни самої методології. У ще більших командах застосування DTP-Indie вимагало б додаткових механізмів координації між кількома підрозробниками кожної колії, що виходить за межі поточного дослідження.

Сформульовано теоретичний підхід до вирішення задачі планування ІТ-проєкту з розробки растрового графічного редактора інді-командою, що базується на чотирьох принципах: просторова декомпозиція проєкту замість темпоральної; декомпозиція за критерієм зв'язності компонентів; формалізований двосторонній зворотний зв'язок між коліями через гейти валідації та запити на корекцію; ресурсна ефективність через мінімізацію адміністративного навантаження. Запропонований підхід принципово відрізняється від описової моделі Water-Scrum-Fall за способом організації потоків роботи (паралельний замість послідовного) та за наявністю формалізованого механізму зворотного зв'язку між компонентами. Узагальнений алгоритм методу включає підготовчу, основну (паралельну) та завершальну фази, охоплюючи повний життєвий цикл проєкту. Формальний опис самого методу DTP-Indie з деталізацією процедур, ролей та критеріїв проводиться у наступному підрозділі.

2.2 Удосконалений гібридний метод планування DTP-Indie

На основі сформульованих у попередньому підрозділі теоретичних принципів пропонується удосконалений гібридний метод планування ІТ-проєкту з розробки растрового графічного редактора, що отримав назву DTP-Indie. Цей підрозділ містить формальний опис методу: визначення компонентів, ролей, артефактів, процедур та критеріїв прийняття рішень. Опис побудовано в

порядку, у якому елементи методу залучаються до роботи протягом життєвого циклу проєкту.

2.2.1 Базова архітектура методу

Метод DTP-Indie організовує процес розробки проєкту у двох паралельних потоках, що далі іменуються коліями. Колія Ядра (Engine Track) керує розробкою архітектурно-критичних компонентів графічного редактора у предиктивному режимі. Колія Інструментарію (Toolset Track) керує розробкою користувацько-орієнтованих компонентів в адаптивному режимі. Обидві колії працюють синхронно протягом усього життєвого циклу проєкту і взаємодіють через два контрольних пункти (гейти) та формалізований механізм запитів на корекцію ядра.

Узагальнена схема методу DTP-Indie з трьома фазами життєвого циклу та паралельною організацією колій наведена на рисунку 2.1.

Як видно з рисунку 2.1, проєкт послідовно проходить три фази, проте на відміну від темпоральних гібридів, основна (паралельна) фаза містить два одночасних потоки роботи. Між коліями розташовані два гейти, що забезпечують контроль якості і архітектурної цілісності, а також канал зворотного зв'язку у вигляді запиту на корекцію ядра, який дозволяє Колії Інструментарію впливати на план Колії Ядра у формалізованому порядку.

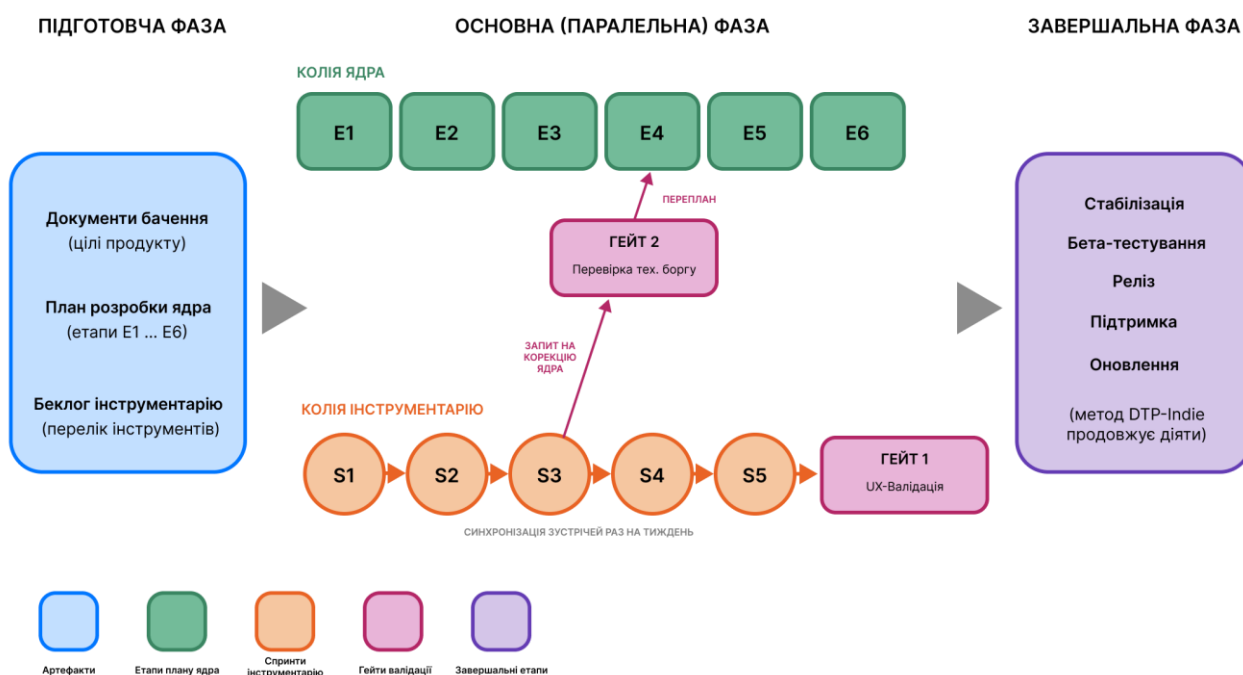


Рисунок 2.1 – Узагальнена схема методу DTP-Indie

2.2.2 Колія Ядра: предиктивне управління архітектурними компонентами

Колія Ядра охоплює всі компоненти системи з високою зв'язністю модулів. Для растрового графічного редактора до них належать математична модель растрової матриці, система виділення пам'яті, базові алгоритми рендерингу, механізм шарів та композиції, операції undo/redo, а також підтримка кольорових просторів і базових форматів файлів.

Принцип проєктування. Компоненти Колії Ядра проєктуються за принципом стабільного публічного API. Внутрішня реалізація алгоритмів може змінюватися протягом проєкту відповідно до запитів на корекцію, проте функції, через які Колія Інструментарію взаємодіє з ядром, повинні залишатися стабільними. Цей принцип, відомий у програмній інженерії як інкапсуляція, забезпечує можливість незалежного розвитку обох колій без взаємного

блокування. У прикладі: інструмент пензля викликає функцію `drawStroke()`; внутрішня реалізація `drawStroke()` може бути повністю переписана у відповідь на запит на корекцію без необхідності змін у коді інструмента.

Структура плану розробки. Розробка Колії Ядра підпорядкована документу під назвою «План розробки ядра» (Engine Roadmap), який формується на підготовчій фазі проєкту і включає 5–7 послідовних етапів. Кожен етап має чітку межу завершення, виражену через перелік функцій публічного API, які мають бути реалізовані. Перехід до наступного етапу можливий лише після завершення поточного. Дозволяється переплан Engine Roadmap у двох випадках: схвалення запиту на корекцію (тимчасове переключення на під-етап) або планове переглянення Roadmap раз на квартал на основі загального прогресу проєкту.

Типовий приклад Engine Roadmap для растрового редактора:

- E1 – базова растрова матриця і canvas-рендерер (3–4 тижні);
- E2 – система виділення пам'яті, undo/redo стек (3 тижні); E3 – Система шарів і базова композиція (4 тижні);
- E4 – розширена композиція, режими накладання, маски (3–4 тижні);
- E5 – кольорові простори, конвертація форматів (2–3 тижні);
- E6 – експорт у PNG/JPG/PSD, оптимізація продуктивності (3 тижні).

Загальна тривалість Engine Track – приблизно 4–5 місяців безперервної роботи одного спеціаліста або 2–3 місяці при розподіленому виконанні.

2.2.3 Колія Інструментарію

Колія Інструментарію охоплює всі компоненти системи з низькою зв'язністю – окремі інструменти малювання, виділення, фільтри, експортні модулі, елементи інтерфейсу. Кожен з цих компонентів спирається на API ядра, але є відносно незалежним від решти компонентів свого рівня.

Структура спринтів. Робота Колії Інструментарію організована короткими ітераціями (спринтами) тривалістю 1–2 тижні. Кожен спринт включає реалізацію одного-двох інструментів та обов'язкове проходження гейту UX-валідації. Спринти не мають жорсткого розкладу, як у класичному Scrum: новий спринт починається після завершення попереднього незалежно від календарної дати. Це дозволяє адаптувати темп розробки до фактичної складності інструментів.

Принцип залежності від API. Колія Інструментарію не може випереджати Колію Ядра. У спринт обираються лише ті інструменти, реалізація яких підтримується поточним станом API ядра. Інструменти, що потребують функцій ядра, які ще не реалізовані, залишаються у беклозі до моменту, коли відповідні етапи Engine Roadmap будуть завершені. Це забезпечує природну синхронізацію темпу обох колій без необхідності штучних бар'єрів.

Беклог інструментарію. Колія Інструментарію спирається на динамічний документ «Беклог інструментарію» (Toolset Backlog), що містить перелік усіх запланованих інструментів з пріоритетами, оцінками складності та позначками залежності від етапів ядра. Беклог є гнучким – пріоритети можуть переглядатися перед кожним новим спринтом на основі результатів попередніх валідацій та зворотного зв'язку від користувачів.

2.2.4 Гейт UX-валідації: трирівневий контроль якості користувацького досвіду

Кожен інструмент, реалізований у Колії Інструментарію, перед інтеграцією в продукт повинен пройти трирівневий гейт UX-валідації. Структура та критерії гейту наведені на рисунку 2.2.

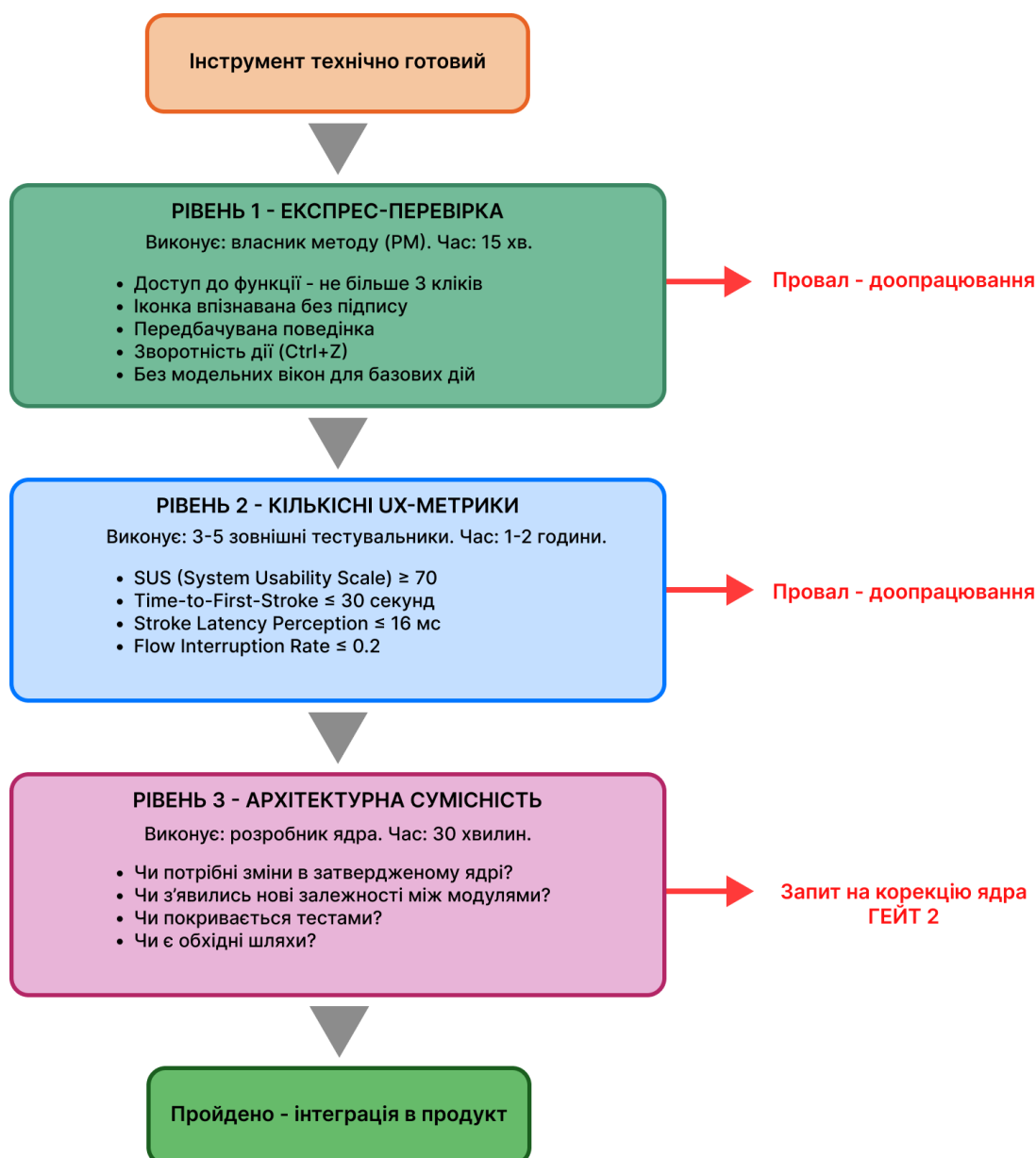


Рисунок 2.2 – Трирівневий гейт UX-валідації

На рисунку 2.2, рівень 1 – експрес-перевірка за чек-листом. Виконується Власником методу за час близько 15 хвилин. Інструмент перевіряється за п'ятьма базовими критеріями ергономіки:

- доступ до функції не більше ніж за 3 кліки від запуску застосунку;
- іконка інструмента впізнавана без супровідного підпису (тестується на членах команди);

- поведінка інструмента передбачувана без прихованих побічних ефектів; зворотність дії через стандартну операцію Ctrl+Z;
- відсутність обов'язкових модальних діалогів для базових операцій.

Інструмент, що не пройшов хоча б один з критеріїв, повертається на доопрацювання без переходу до наступних рівнів.

Рівень 2 на рисунку 2.2 – кількісні UX-метрики. Виконується із залученням 3–5 зовнішніх тестерів за 1–2 години. Інструмент проходить тестування за чотирма формалізованими метриками з визначеними пороговими значеннями. Першою метрикою є System Usability Scale (SUS), стандартний інструмент кількісної оцінки юзабіліті, запропонований Дж. Бруком у 1996 році, що передбачає опитування з 10 питань і дає підсумковий бал від 0 до 100; пороговим значенням є $SUS \geq 70$ (індустріальна норма good UX). Другою метрикою є Time-to-First-Stroke (TFS) (час до першого мазка) – час від запуску застосунку до першої успішної взаємодії новачка з інструментом, з пороговим значенням ≤ 30 секунд.

Третьою метрикою є Stroke Latency Perception SLP – суб'єктивне відчуття затримки між дією користувача і реакцією системи, з пороговим значенням ≤ 16 мілісекунд (один кадр на 60 FPS).

Четвертою метрикою є Flow Interruption Rate (FIR) – частота переривань творчого процесу через незрозумілі або непередбачувані реакції системи, з пороговим значенням ≤ 0.2 переривань на хвилину.

Рівень 3 (рисунок 2.2) – перевірка архітектурної сумісності. Виконується Розробником ядра за час близько 30 хвилин. На цьому рівні аналізується вплив інструмента на архітектурну цілісність ядра за чотирма критеріями: чи потребує інструмент змін у затвердженому API ядра; чи з'являються нові залежності між модулями; чи покривається інструмент автоматизованими тестами; чи є у його реалізації обхідні шляхи, що тимчасово компенсують відсутність потрібного функціоналу в ядрі. Якщо інструмент потребує змін у ядрі, спрацьовує механізм запиту на корекцію, описаний у наступному підпункті.

Інструменти, що пройшли всі три рівні валідації, інтегруються в основну гілку продукту. Інструменти, що провалилися на Рівні 1 або 2, повертаються розробнику інструментарію на доопрацювання. Інструменти, що провалилися на Рівні 3, ініціюють процедуру запиту на корекцію ядра.

2.2.5 Запит на корекцію ядра і Гейт перевірки технічного боргу

Запит на корекцію ядра (англ. Architectural Change Request, ACR) – формалізована процедура, через яку Колія Інструментарію може ініціювати зміну в плані розробки ядра. Це ключовий інноваційний механізм методу DTP-Indie, відсутній в темпоральних гібридних методах. Процес обробки запиту наведено на рисунку 2.3.

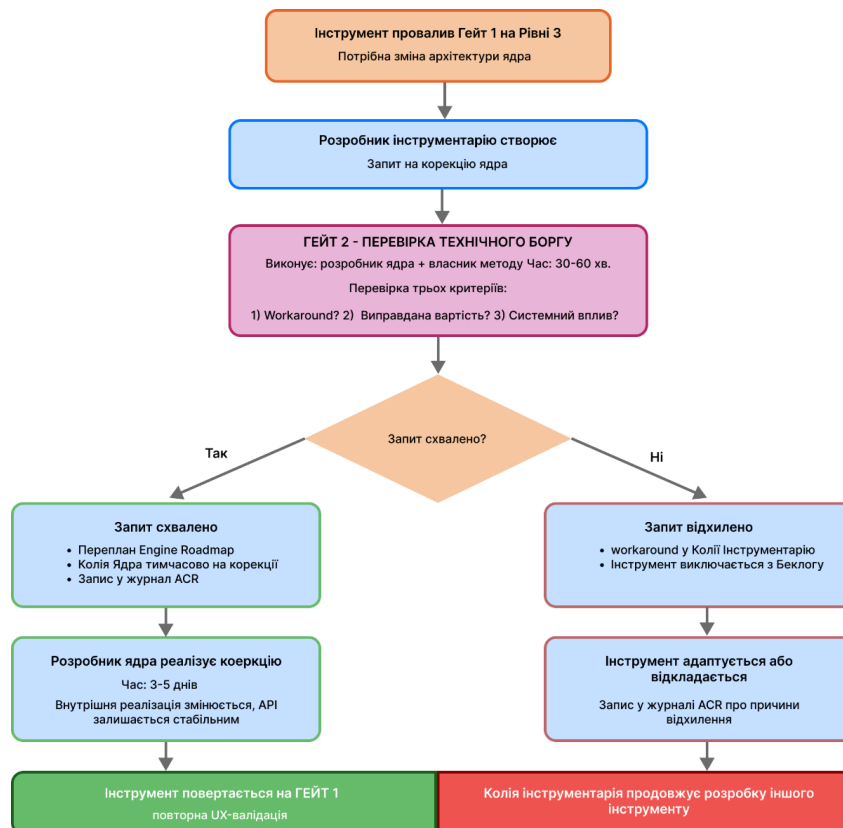


Рисунок 2.3 – Процес обробки запиту на корекцію ядра

Розробник інструментарію формує запит за стандартизованим шаблоном, що включає п'ять полів: коротку назву зміни, що потрібна; обґрунтування (який інструмент і яку UX-метрику не може задовольнити без зміни); альтернативи (чи можливий обхідний шлях у Колії Інструментарію, і чому він небажаний); очікуваний обсяг роботи у днях; вплив на Engine Roadmap (які етапи будуть зачеплені або відкладені). Заповнення шаблону займає 10–15 хвилин і не вимагає окремої документації.

Гейт перевірки технічного боргу. Запит проходить процедуру оцінки, яку виконують спільно Розробник ядра і Власник методу. Перевіряються три критерії: чи дійсно неможливо реалізувати інструмент через обхідне рішення без зміни ядра; чи виправдана зміна співвідношенням вартості робіт до отриманої функціональності; чи має зміна системний вплив на майбутні інструменти, або стосується лише поточного. Системні зміни мають вищий пріоритет схвалення, ніж локальні. Тривалість оцінки – 30–60 хвилин.

Прийняття рішення. У разі схвалення запиту в Engine Roadmap додається тимчасовий під-етап, на період якого Колія Ядра переключається з основного плану на реалізацію корекції. Зазвичай це 3–7 днів роботи. Після завершення корекції робота за основним планом ядра відновлюється. У разі відхилення запиту інструмент або реалізується з обхідним рішенням (з відповідним записом у журналі), або відкладається у беклозі до моменту, коли архітектура ядра природним чином дозволить його реалізацію без додаткових витрат.

Журнал ACR. Усі запити, незалежно від результату, фіксуються у журналі ACR. Це проста таблиця, що містить дату запиту, назву інструмента-ініціатора, рішення (схвалено/відхилено), фактичний час реалізації та коротку рефлексію щодо точності початкової оцінки. Журнал є цінним артефактом для подальшої калібрування оцінок та аналізу типових архітектурних розривів.

2.2.6 Ролі, артефакти та церемонії методу

Метод DTP-Indie свідомо мінімізує адміністративне навантаження на команду через обмежену кількість ролей, артефактів та регулярних церемоній. Загальна структура організаційного компонента методу наведена на рисунку 2.4.

Ролі. Метод передбачає три функціональні ролі. Власник методу (Method Owner) поєднує функції менеджера проєкту і технічного лідера: координує колії, виконує гейти, приймає рішення на основі результатів валідації. Розробник ядра (Engine Developer) реалізує план розробки ядра, оцінює технічну сторону запитів на корекцію, відповідає за стабільність API. Розробник інструментарію (Toolset Developer) реалізує конкретні інструменти, проводить попереднє тестування, ініціює запити при потребі. У командах з 2–3 осіб допускається поєднання ролей: одна особа може виступати як Власник методу та Розробник ядра одночасно. У більших командах ролі розподіляються між окремими спеціалістами.

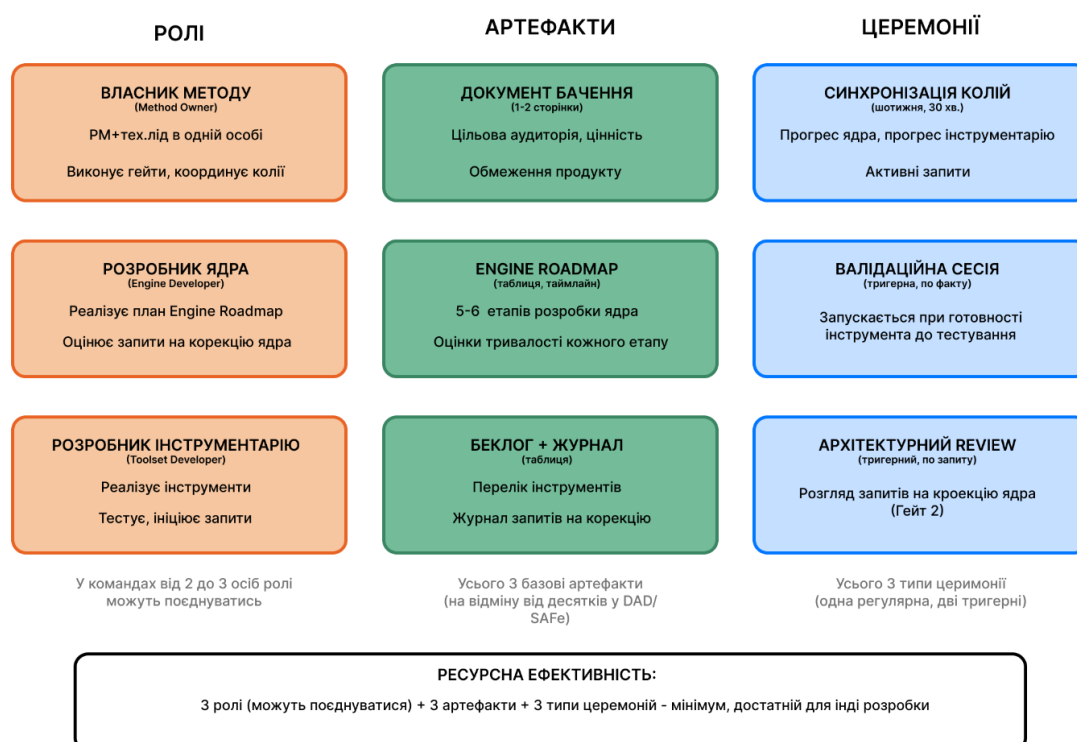


Рисунок 2.4 – Ролі, артефакти та церемонії методу DTP-Indie

Метод використовує три базові документи. Документ бачення (1–2 сторінки) визначає цільову аудиторію продукту, ключову цінність і обмеження; формується на підготовчій фазі і оновлюється рідко. Engine Roadmap (1 сторінка) містить етапи розробки ядра з оцінками тривалості; оновлюється при схваленні запитів на корекцію та плановому квартальному перегляді. Беклог інструментарію з журналом ACR (динамічна таблиця) поєднує перелік запланованих інструментів і журнал усіх запитів на корекцію; оновлюється у щоденному режимі. На відміну від корпоративних методів, що передбачають десятки документів (DAD, SAFe), DTP-Indie свідомо обмежується трьома базовими артефактами.

Церемонії. Метод передбачає три типи церемоній, з яких лише одна є регулярною. Синхронізаційна зустріч колій проводиться раз на тиждень тривалістю 30 хвилин і включає звіт про прогрес обох колій, обговорення активних запитів на корекцію та узгодження пріоритетів наступного тижня. Валідаційна сесія є тригерною – запускається тоді, коли інструмент технічно готовий до перевірки на гейті UX-валідації. Архітектурний review теж є тригерним – запускається при надходженні запиту на корекцію для проведення Гейту перевірки технічного боргу. Загалом регулярна частина адміністрування займає 30 хвилин на тиждень, що принципово менше за вимоги корпоративних гібридних методів.

2.2.7 Завершальна фаза і життєвий цикл після релізу

Після того як основна функціональність продукту реалізована і пройшла валідацію, проєкт переходить у завершальну фазу. На цьому етапі обидві колії змінюють режим роботи: Колія Ядра фокусується на оптимізації продуктивності,

профілюванні і виправленні критичних дефектів, виявлених при інтеграційному тестуванні; Колія Інструментарію припиняє додавати нові інструменти, натомість працюючи над поліпшенням існуючих за результатами бета-тестування.

Розглянемо етапи завершальної фази:

- стабілізація – період активного виправлення дефектів і регресійного тестування (типова тривалість 2–4 тижні);
- бета-тестування – обмежений публічний випуск для збору зворотного зв'язку від реальних користувачів (4–6 тижнів);
- реліз – повний публічний випуск продукту;
- підтримка і оновлення – постійна фаза, у якій метод DTP-Indie продовжує діяти у скороченому режимі.

Метод DTP-Indie у режимі підтримки. Після релізу метод не припиняє роботу, але адаптується до нових умов. Великі архітектурні оновлення (підтримка нової операційної системи, перехід на нову версію графічного API) проходять як окремі під-проекти Колії Ядра з власним планом і гейтами. Невеликі поліпшення інтерфейсу і нові інструменти продовжують надходити через Колію Інструментарію зі стандартною UX-валідацією. Виправлення критичних дефектів виконуються поза коліями за процедурою швидкого виправлення без проходження повних гейтів. Така структура дозволяє методу залишатися релевантним протягом усього життєвого циклу продукту, а не тільки на етапі його початкової розробки.

2.2.8 Сфера застосування методу

Метод DTP-Indie спроектований передусім для розробки растрових графічних редакторів інді-командами, проте його структурні принципи

застосовні до ширшого класу проєктів. Сферу застосування методу можна охарактеризувати за двома вимірами – масштабом команди і типом продукту.

За масштабом команди, оптимальний діапазон – 2–7 осіб, у якому метод працює без модифікацій за рахунок поєднання ролей. У командах 8–15 осіб метод залишається застосовним, але ролі розподіляються між окремими спеціалістами і документація запитів на корекцію стає більш формалізованою. У командах 15–50 осіб метод вимагає модифікації шляхом розбиття колій на підкоманди з окремим координатором; ця розширена версія методу позначена як DTP-Studio і виходить за межі поточного дослідження. У великих корпоративних командах (понад 50 осіб) метод не масштабується і доцільно використовувати DAD або SAFe.

За типом продукту – метод ефективний для будь-яких продуктів з двокомпонентною структурою «обчислювально-критичне ядро з API + користувацький шар, що споживає API». До таких продуктів належать векторні графічні редактори, аудіо- та відео-редактори, тривимірні графічні застосунки, ігрові рушії з власним редактором, інтегровані середовища розробки програмного забезпечення. Для продуктів без такої структури – наприклад, типових веб-застосунків – метод дає менше переваг порівняно зі звичайним Scrum, оскільки в них відсутнє стабільне ядро з високою зв'язністю, і паралельна організація колій втрачає сенс.

Сформульовано формальний опис удосконаленого гібридного методу DTP-Indie, що включає базову архітектуру з двома паралельними коліями, структуру передбачуваного управління Колією Ядра з принципом стабільного API, структуру адаптивного управління Колією Інструментарію з прив'язкою до API ядра, тривірневий гейт UX-валідації з кількісними метриками SUS, TFS, SLP та Flow Interruption Rate, формалізовану процедуру запитів на корекцію ядра з відповідним гейтом перевірки технічного боргу, мінімалістичну організаційну структуру з трьох ролей, трьох артефактів та трьох типів церемоній. Описано також завершальну фазу і режим підтримки продукту після релізу. Визначено

сферу застосування методу за масштабом команди та типом продукту. Розгляд ризиків застосування методу та аналіз його обмежень проводиться у наступному підрозділі.

2.3 Управління ризиками та обмеження методу DTP-Indie

Будь-який метод управління ІТ-проєктами має внутрішні ризики застосування та обмеження сфери дії, об'єктивне визнання яких є необхідною умовою академічної зрілості методологічної розробки. Сформульований у попередньому підрозділі метод DTP-Indie не є винятком: попри обґрунтовані переваги порівняно з існуючими підходами, його практичне застосування пов'язане з низкою специфічних ризиків, а сфера ефективного використання має чітко визначені межі.

2.3.1 Класифікація ризиків методу

Аналіз структурних характеристик методу DTP-Indie та умов його застосування у середовищі інді-розробки дозволяє виділити шість типових ризиків, що мають різну природу і потребують різних підходів до управління. Узагальнена матриця цих ризиків з оцінкою ймовірності та наслідків наведена на рисунку 2.5.

Матриця побудована за стандартною методикою управління ризиками: горизонтальна вісь відображає потенційні наслідки матеріалізації ризику для проєкту, вертикальна – оцінку ймовірності його настання у типовому проєкті інді-розробки растрового редактора. Кольорова градація від зеленого до

червоного відповідає рівню сукупної загрози ризику [16]. Розгляд кожного з виявлених ризиків з обґрунтуванням оцінок проводиться нижче.

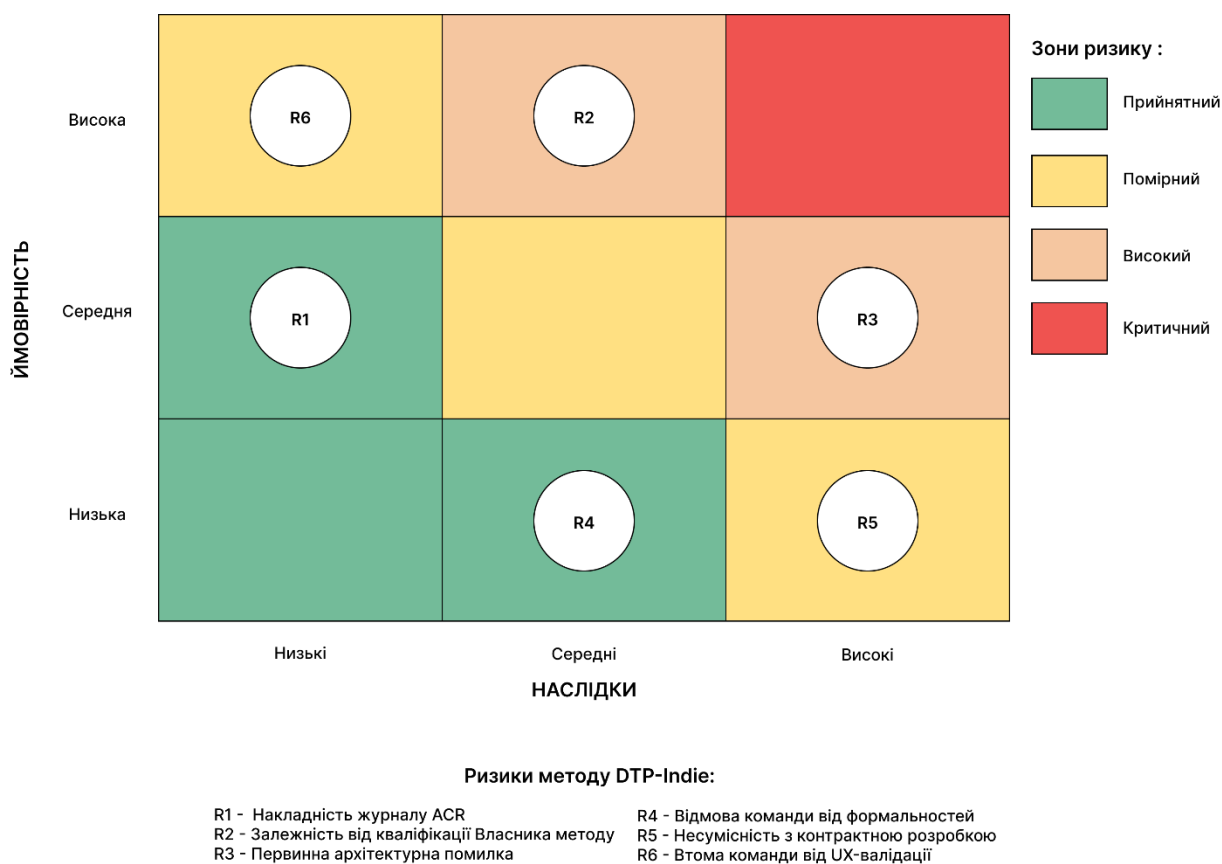


Рисунок 2.5 – Матриця ризиків методу DTP-Indie

Ризик R1 – накладні витрати на ведення журналу запитів на корекцію. Природа ризику: формалізована процедура подачі та реєстрації запитів на корекцію вимагає від команди витрат часу на заповнення шаблону, проведення архітектурного review та оновлення журналу. У дуже малих командах (2 особи) ці витрати можуть здаватися надмірними порівняно з можливістю усного узгодження архітектурних змін. Оцінка ризику: помірна ймовірність, низькі наслідки. Захід мітигації: шаблон запиту свідомо обмежено п'ятьма полями і часом заповнення 10-15 хвилин, журнал ведеться в простій таблиці без спеціалізованих інструментів, для команд 2 осіб дозволяється спрощений усно-

задокументований формат, у якому архітектурні зміни обговорюються усно, але обов'язково фіксуються коротким записом у журналі.

Ризик R2 – залежність ефективності методу від кваліфікації Власника методу. Природа ризику: на відміну від директивних методів з жорстким центральним планом (каскадна модель, V-подібна модель), DTP-Indie передбачає прийняття рішень на гейтах валідації Власником методу, який поєднує функції менеджера і технічного лідера. Якщо ця особа має недостатній досвід у розробці графічного програмного забезпечення, рішення про схвалення або відхилення запитів на корекцію можуть бути неточними, що призводить до накопичення необґрунтованих архітектурних змін або, навпаки, до блокування потрібних змін. Оцінка ризику: висока ймовірність, середні наслідки. Захід мітигації: метод явно встановлює мінімальні кваліфікаційні вимоги до Власника методу – мінімум 2 роки досвіду в розробці графічного або мультимедійного програмного забезпечення; рекомендується періодичне коригування рішень на основі ретроспективного аналізу журналу запитів на корекцію кожні 3-4 місяці; у разі систематичних помилок передбачається можливість залучення зовнішнього консультанта-архітектора для разової валідації плану розробки ядра.

Ризик R3 – первинна архітектурна помилка на підготовчій фазі. Природа ризику: метод DTP-Indie дозволяє коригувати архітектуру ядра через формальні запити, проте не призначений для виявлення фундаментальних помилок початкового вибору технологічного стеку або базових структур даних. Якщо на підготовчій фазі обрано принципово непридатний фреймворк або базова модель растрової матриці спроектована з критичною помилкою, перші кілька запитів на корекцію стануть індикатором проблеми, але до того часу буде втрачено кілька тижнів роботи. Оцінка ризику: середня ймовірність, високі наслідки. Захід мітигації: метод включає обов'язкову рекомендацію створення прототипу-перевірки (proof of concept) тривалістю 1–2 тижні перед затвердженням плану розробки ядра; цей прототип реалізує мінімальний робочий цикл – створення

canvas, малювання базового штриха і збереження результату – на цільовому технологічному стеку, що дозволяє валідувати архітектурні припущення до повного запуску проєкту.

Ризик R4 – неформальна відмова команди від процедур методу. Природа ризику: у малих командах з тісними особистими стосунками існує спокуса обійти формальні процедури методу – пропустити рівень UX-валідації, не оформити запит на корекцію, провести архітектурний review усно без запису у журналі. Така деградація методу повертає команду до неформального хаос-підходу і нівелює усі переваги формалізації. Оцінка ризику: низька ймовірність (за умови усвідомленого вибору методу), середні наслідки. Захід мітигації: метод проєктувався з принципом мінімальних адміністративних витрат саме для зниження спокуси його обходити, у періодичних ретроспективах рекомендується явно оцінювати дотримання процедур, у разі виявлення систематичних відмов від формальностей рекомендується перегляд вибору методу – можливо, для конкретної команди більш доцільним є чистий Scrum або Kanban.

Ризик R5 – несумісність з контрактною моделлю розробки. Природа ризику: метод DTP-Indie передбачає динамічне переплан Engine Roadmap у відповідь на запити на корекцію, що несумісно з контрактними угодами на розробку з фіксованою специфікацією і незмінним планом. Якщо інді-команда виконує проєкт за контрактом «під ключ» з повністю заздалегідь обумовленою функціональністю, формальне проходження запитів на корекцію може створити юридичні проблеми у відносинах із замовником. Оцінка ризику: низька ймовірність (інді-команди рідко працюють за такою моделлю), високі наслідки у разі настання. Захід мітигації: метод DTP-Indie не рекомендується для проєктів з фіксованою специфікацією; для контрактної розробки доцільно використовувати V-подібну модель або класичну каскадну модель з прозорою фіксацією плану; у випадку гібридних угод (фіксований обсяг ядра, гнучкий

обсяг інструментарію) метод може бути адаптований через окремий формат контракту, що передбачає зміни в плані Колії Ядра як офіційну поправку.

Ризик R6 – втома команди від регулярної UX-валідації. Природа ризику: трирівневий гейт UX-валідації, що проходить кожен інструмент перед інтеграцією, є ресурсо-витратною процедурою; із зростанням кількості інструментів у продукті команда може почати сприймати валідацію як рутинну формальність, що призведе до зниження якості перевірок та накопичення прихованих UX-проблем. Оцінка ризику: висока ймовірність (особливо після 6–9 місяців проєкту), низькі наслідки. Захід мітигації: метод передбачає ротацію тестерів на Рівні 2 – в ідеалі, кожні 2–3 місяці склад зовнішніх тестерів має оновлюватися для збереження «свіжого ока»; для постійних команд рекомендується чергування ролей – Розробник інструментарію періодично проходить тренінг з UX-аналізу та сам стає тестером для інструментів, реалізованих іншими членами команди; для критично важливих інструментів проводиться розширене тестування з 7–10 учасниками замість стандартних 3–5.

2.3.2 Структурні обмеження методу

Окремо від ризиків, що пов'язані зі специфікою застосування методу у конкретних проєктах, доцільно виділити структурні обмеження, які впливають з самої архітектури методу і визначають класи задач, для яких DTP-Indie принципово не призначений. Чесне визнання цих обмежень підвищує методологічну зрілість дослідження і допомагає потенційним користувачам коректно ідентифікувати застосовність методу до своїх проєктів.

Обмеження 1 – метод не масштабується на великі команди без модифікацій. DTP-Indie проєктувався для команд 2–7 осіб, у яких можлива безпосередня неформальна комунікація між учасниками обох колій. У командах

від 15 осіб і більше метод вимагає введення додаткових механізмів координації між під-командами кожної колії, виокремлення ролі координатора, формалізації каналів передачі інформації – тобто фактично переходить у категорію корпоративних методологій (DAD, SAFe). Спроба механічно застосувати DTP-Indie до великих команд призведе до втрати ресурсної ефективності без отримання відповідних організаційних переваг.

Обмеження 2 – метод не призначений для продуктів без чіткого розділення ядра і UI. Принцип просторової декомпозиції за зв'язністю компонентів передбачає наявність у системі обчислювально-критичного ядра з API та користувацького шару, що споживає це API. Для класів продуктів, де така структура відсутня – типові веб-застосунки на основі REST API, бізнес-додатки з простою логікою – метод не дає переваг порівняно зі звичайним Scrum, оскільки декомпонувати таку систему на дві якісно різні колії неможливо.

Обмеження 3 – метод не вирішує проблему вибору технологічного стеку. DTP-Indie припускає, що базові технологічні рішення (мова програмування, графічний фреймворк, цільові платформи) приймаються на підготовчій фазі і не переглядаються протягом проєкту. Якщо проєкт стикається з фундаментальною технологічною проблемою (наприклад, обраний фреймворк не підтримує цільову платформу), метод не пропонує процедури для зміни стеку – у такій ситуації проєкт фактично починається спочатку. Це обмеження є типовим для усіх методів управління IT-проєктами і не специфічне для DTP-Indie.

Обмеження 4 – Метод залежить від зовнішніх інструментів вимірювання UX-метрик. Кількісні метрики Рівня 2 гейту UX-валідації (SUS, TFS, SLP, FIR) потребують доступу до зовнішніх тестерів та інструментів вимірювання. Для команд, що не мають доступу до тестувальної аудиторії або не можуть забезпечити інструментальне вимірювання затримок (наприклад, через відсутність профілювальних інструментів), повноцінне застосування Рівня 2 ускладнене. Метод передбачає у такому випадку розширення Рівня 1 – більш

детальний чек-лист експрес-перевірки, який частково компенсує відсутність кількісних метрик, але не замінює їх повністю.

2.3.3 Узагальнення: рекомендації щодо застосування методу

На основі проведеного аналізу ризиків та обмежень можна сформулювати критерії, за яких метод DTP-Indie рекомендований до застосування, не рекомендований, або застосовний з модифікаціями. Узагальнення цих критеріїв наведено у таблиці 2.1.

Таблиця 2.1 – Критерії застосовності методу DTP-Indie

Рекомендований	Застосовний з модифікаціями	Не рекомендований
Команди 2–7 осіб	Команди 8–15 осіб	Команди понад 15 осіб (доцільно DAD/SAFe)
Растрові графічні редактори	Векторні редактори, аудіо/відео-редактори, ігрові рушії з редактором	Веб-застосунки, бізнес-додатки без обчислювального ядра
Самостійна продуктова розробка	Гібридні контракти з фіксованим ядром і гнучким UI	Контрактна розробка під ключ з фіксованою специфікацією
Власник методу з 2+ роками досвіду	Команди з керівником без спеціалізованого досвіду + зовнішня консультація	Команди без жодного учасника з досвідом

Подана таблиця призначена не лише для академічного представлення обмежень методу, а й для практичної допомоги командам, що розглядають застосування DTP-Indie у своїх проєктах. Чітка ідентифікація проєкту в одній з трьох категорій таблиці дозволяє свідомо приймати рішення про доцільність

використання методу без ризику отримати негативний досвід через неправильний вибір.

Проведено систематичний аналіз ризиків та обмежень методу DTP-Indie. Виявлено шість типових ризиків застосування методу з оцінкою ймовірності та наслідків кожного, а також сформульовано конкретні заходи мітигації для кожного ризику. Окремо проаналізовано чотири структурних обмеження методу, що визначають класи задач, для яких він не призначений. Сформульовано узагальнені критерії застосовності методу за трьома параметрами – масштабом команди, типом продукту та моделлю розробки. Чесне визнання внутрішніх обмежень методу та їх явне документування підвищує методологічну зрілість дослідження і дозволяє потенційним користувачам коректно ідентифікувати застосовність методу до конкретних проєктів. Розгляд кількісних метрик ефективності методу проводиться у наступному підрозділі.

2.4 Система метрик ефективності методу DTP-Indie

Будь-який метод управління IT-проєктами може претендувати на наукову обґрунтованість лише за умови, що його ефективність піддається кількісному вимірюванню [16]. Якісна оцінка методу через експертне судження або суб'єктивні відчуття команди не є достатньою для верифікації, оскільки не дозволяє відтворити результат, порівняти різні методи між собою або виявити чинники, що впливають на якість управління. Цей підрозділ присвячений формуванню системи метрик, що дозволяють кількісно оцінити ефективність методу DTP-Indie у трьох вимірах: точність планування процесу, якість отриманого продукту та інтегральна ефективність методу як цілого.

2.4.1 Структура системи метрик

Запропонована система метрик базується на принципі дворівневого вимірювання: окремо оцінюються характеристики процесу управління (як працює метод) та характеристики продукту, отриманого в результаті (що отримано на виході). Ці дві категорії доповнюють одна одну: метод може бути добре організованим, але призвести до низькоякісного продукту, або навпаки, дати якісний продукт ціною хаотичного процесу. Лише одночасний контроль обох категорій дозволяє коректно оцінити ефективність методу.

Загальна структура системи метрик з пороговими значеннями для кожного показника наведена на рисунку 2.6.



Рисунок 2.6 – Система метрик ефективності методу DTP-Indie

Метрики процесу включають чотири показники: точність оцінки тривалості (Estimation Accuracy, EA), частку повторних циклів валідації (Rework Cycle Rate, RCR), індекс технічного боргу (Technical Debt Index, TDI) та частку схвалених запитів на корекцію (ACR Approval Rate, AAR). Метрики продукту також включають чотири показники, що були попередньо введені у підрозділі

2.2 для гейту UX-валідації: SUS, TFS, SLP та Flow Interruption Rate (FIR). Деталізація формул, методів обчислення та порогових значень для кожної метрики проводиться нижче [16].

2.4.2 Метрики процесу управління

Точність оцінки тривалості ЕА. Метрика характеризує, наскільки точно команда оцінює тривалість етапів плану розробки ядра на підготовчій фазі. Низька точність оцінки свідчить про недостатнє розуміння технічної складності компонентів або про відсутність калібрування на основі попередніх проєктів. Метрика обчислюється як середнє абсолютне відхилення фактичної тривалості від планової за всіма завершеними етапами:

$$EA = \frac{1}{n} \sum_{i=1}^n \frac{|T_{\text{факт},i} - T_{\text{план},i}|}{T_{\text{план},i}} \cdot 100\%, \quad (2.1)$$

де n – кількість завершених етапів плану розробки ядра;

$T_{\text{факт},i}$ – фактична тривалість i -го етапу;

$T_{\text{план},i}$ – планова тривалість i -го етапу.

Метрика вимірюється у відсотках і має пороговим значенням $EA < 20\%$, що відповідає індустріальній нормі для проєктів з помірною технологічною невизначеністю. Перевищення порогу свідчить про необхідність калібрування методики оцінки – наприклад, через залучення зовнішнього технічного експерта на підготовчій фазі або використання історичних даних з попередніх проєктів команди.

Частка повторних циклів валідації RCR. Метрика характеризує, як часто інструменти повертаються на доопрацювання після провалу одного з рівнів гейту UX-валідації. Висока частка повторних циклів свідчить про недостатню якість

попереднього тестування Розробником інструментарію або про неточно сформульовані вимоги до інструментів. Формула обчислення:

$$RCR = \frac{N_{rework}}{N_{total}}, \quad (2.2)$$

де N_{rework} – кількість випадків повернення інструментів на доопрацювання за весь період спостереження;

N_{total} – загальна кількість запусків гейту UX-валідації за той же період. Метрика приймає значення від 0 до 1 і має пороговим значенням $RCR < 0.30$. Це означає, що не більше 30% інструментів повертається на доопрацювання – інакше повторні цикли стають домінуючим режимом роботи команди, що нівелює переваги адаптивного управління Колією Інструментарію.

Індекс технічного боргу TDI. Метрика характеризує накопичення компромісних рішень та обхідних шляхів у кодовій базі продукту. Технічний борг є природним наслідком будь-якої адаптивної розробки, проте його контрольоване накопичення є необхідною умовою стабільності методу. Метрика обчислюється як кількість позначок-маркерів обхідних рішень у коді (TODO, FIXME, HACK, XXX) на одиницю обсягу кодової бази:

$$TDI = \frac{N_{TODO} + N_{FIXME} + N_{HACK} + N_{workaround}}{LOC / 1000}, \quad (2.3)$$

де N_{TODO} , N_{FIXME} , N_{HACK} – кількість відповідних маркерів у коментарях

$N_{workaround}$ – кількість задокументованих обхідних рішень у журналі ACR;

LOC (Lines of Code) – загальна кількість значущих рядків коду продукту.

Метрика виражається в одиницях боргу на 1000 LOC і має пороговим значенням $TDI < 5$. Перевищення порогу свідчить про систематичну відмову Гейту перевірки технічного боргу від схвалення необхідних запитів на корекцію

– у такій ситуації Колія Інструментарію змушена застосовувати обхідні рішення замість адекватної корекції ядра.

Частка схвалених запитів на корекцію (ACR Approval Rate, AAR). Метрика характеризує, наскільки точно команда формулює запити на корекцію ядра – чи обґрунтовані вони реальною необхідністю, чи виникають через помилки розробника інструментарію, які можна було б усунути на місці. Формула обчислення:

$$AAR = \frac{N_{approved}}{N_{submitted}}, \quad (2.4)$$

де $N_{approved}$ – кількість схвалених запитів;

$N_{submitted}$ – загальна кількість поданих запитів.

Метрика приймає значення від 0 до 1 і має цільовим діапазоном AAR від 0.50 до 0.80. На відміну від попередніх метрик, для AAR небажані як надто низькі, так і надто високі значення. Низька частка схвалень ($AAR < 0.50$) свідчить про те, що Розробник інструментарію формує надто багато необґрунтованих запитів – потрібне додаткове навчання або наставництво. Надто висока частка схвалень ($AAR > 0.80$) свідчить про недостатню критичність Гейту перевірки технічного боргу – Розробник ядра приймає всі запити поспіль, що порушує дисципліну архітектурного контролю.

2.4.3 Метрики якості продукту

Чотири метрики продукту, що вже згадувалися у складі гейту UX-валідації (підрозділ 2.2.4), служать одночасно інструментом контролю окремих інструментів та інтегральним показником якості усього продукту на

завершальній фазі проєкту. Нижче наводиться їх формальний опис з формулами обчислення для агрегації по продукту в цілому.

SUS є стандартизованою методикою кількісної оцінки юзабіліті інтерфейсів, запропонованою Дж. Бруком у 1996 році. Респондент відповідає на 10 стандартних запитань за п'ятибальною шкалою Лайкерта (від 1 – категорично не згоден, до 5 – повністю згоден). Підсумковий бал обчислюється за формулою:

$$SUS = 2,5 \cdot [\sum_{i \in odd} (Q_i - 1) + \sum_{i \in even} (5 - Q_i)], \quad (2.5)$$

де Q_i – оцінка респондента на i -те запитання, підсумовування по непарних запитаннях (odd) додає $(Q_i - 1)$, по парних (even) – $(5 - Q_i)$, результат масштабується множителем 2.5 до діапазону 0-100.

Для агрегації SUS по всьому продукту обчислюється середнє арифметичне SUS-балів усіх інструментів, протестованих на Рівні 2 гейту UX-валідації. Цільове значення – $SUS \geq 70$, що відповідає індустріальній категорії «good usability» згідно з градацією Бангора, Кортума та Міллера (2009).

TFS вимірює час від запуску застосунку до моменту, коли новий користувач без попередньої інструкції здійснює перший успішний мазок пензля на полотні. Метрика характеризує доступність базового функціоналу для нових користувачів – критичну характеристику растрового редактора, орієнтованого на низький поріг входження. Метрика безпосередньо вимірюється секундоміром під час тестування і агрегується по продукту як медіана значень для всіх протестованих новачків:

$$TFS_{product} = median \{TFS_i, i = 1..n\}, \quad (2.6)$$

де TFS_i – час до першого мазка для i -го тестера,

n – кількість тестерів-новачків.

Цільове значення $TFS \leq 30$ секунд.

Використання медіани замість середнього значення обумовлене стійкістю медіани до викидів, які можуть виникати при тестуванні через випадкові фактори (наприклад, проблеми з обладнанням у конкретного тестера).

SLP характеризує відчутну користувачем затримку між дією (рухом пензля) та реакцією системи (появою мазка на полотні). Метрика має комбіновану природу: об'єктивну (інструментально вимірний час обробки) та суб'єктивну (оцінка тестером відчуття плавності). Об'єктивна складова обчислюється як середній час між подією введення (input event) та подією завершення відображення (frame render):

$$SLP_{obj} = \frac{1}{n} \sum_{i=1}^n (T_{render,i} - T_{input,i}), \quad (2.7)$$

де $T_{input,i}$ – момент події введення;

$T_{render,i}$ – момент завершення відображення відповідного кадру;

n – кількість вимірних подій.

Цільове значення $SLP \leq 16$ мілісекунд, що відповідає одному кадру при частоті оновлення екрана 60 Гц. Перевищення цього порогу робить затримку відчутною для людського ока і призводить до сприйняття інструмента як «підвисаючого». Суб'єктивна складова метрики визначається через опитування тестерів за п'ятибальною шкалою плавності і служить контрольним показником на випадок, коли об'єктивне вимірювання технічно неможливе.

FIR характеризує частоту переривань творчого процесу користувача через незрозумілі поведінки інтерфейсу, неочікувані модальні вікна, помилки або потребу шукати функцію. Метрика обчислюється як кількість переривань на одиницю часу активної роботи з продуктом:

$$FIR = \frac{N_{interrupt}}{T_{active}}, \quad (2.8)$$

де $N_{interrupt}$ – кількість зафіксованих переривань творчого процесу за період спостереження;

T_{active} – тривалість активної роботи з продуктом у хвилинах.

Тестер протягом контрольної сесії (тривалістю 15–30 хвилин) виконує типові творчі завдання і самостійно фіксує моменти, коли його увага переключається з творчого процесу на боротьбу з інтерфейсом. Цільове значення $FIR \leq 0.20$ переривань на хвилину, що еквівалентно одному перериванню на 5 хвилин активної роботи.

2.4.4 Інтегральний показник ефективності методу

Інтегральний показник ефективності E пропонується обчислювати як зважену суму нормалізованих значень окремих метрик, де нормалізація приводить кожен метрику до діапазону $[0, 1]$ (де 1 відповідає ідеальному стану, 0 – катастрофічному):

$$E = w_1 \cdot EA_{norm} + w_2 \cdot (1 - RCR) + w_3 \cdot SUS_{norm} + w_4 \cdot (1 - FIR_{norm}) \quad (2.9)$$

Ваги w_1, w_2, w_3, w_4 підбираються залежно від пріоритетів проєкту і повинні задовольняти умові $w_1 + w_2 + w_3 + w_4 = 1$.

Для типового проєкту інді-розробки растрового редактора рекомендуються такі значення вагів: $w_1 = 0.20$ (точність планування), $w_2 = 0.20$ (стабільність процесу), $w_3 = 0.35$ (юзабіліті як ключова цінність продукту), $w_4 = 0.25$ (плавність роботи).

2.4.5 Узагальнена характеристика метрик

Систематизована характеристика всіх восьми основних метрик з формулами, пороговими значеннями та інтерпретацією наведена у таблиці 2.2.

Таблиця 2.2 – Метрики ефективності методу DTP-Indie

Позна-чення	Назва	Формула	Поріг	Інтерпретація
EA	Точність оцінки тривалості	(2.1)	< 20%	Менше – точніше планується час етапів ядра
RCR	Частка повторних циклів валідації	(2.2)	< 0.30	Менше – рідше інструменти повертаються на доопрацювання
TDI	Індекс технічного боргу	(2.3)	< 5	Менше – менший борг на 1000 рядків коду
AAR	Частка схвалених запитів	(2.4)	0.50–0.80	В діапазоні – баланс архітектурного контролю
SUS	System Usability Scale	(2.5)	≥ 70	Більше – кращий юзабіліті продукту
TFS	Time-to-First-Stroke	(2.6)	≤ 30 с	Менше – нижчий поріг входження для новачків
SLP	Stroke Latency Perception	(2.7)	≤ 16 мс	Менше – плавніша реакція на дії користувача
FIR	Flow Interruption Rate	(2.8)	≤ 0.20	Менше – менше переривань творчого процесу
E	Інтегральний показник ефективності	(2.9)	≥ 0.80	Більше – вища загальна ефективність методу

Запропонована система метрик не претендує на вичерпність – на практиці окремі команди можуть доповнювати її специфічними показниками для своїх проєктів (наприклад, метриками продуктивності для оптимізаційно-критичних інструментів або метриками доступності для користувачів з обмеженнями).

Однак запропонований набір з восьми базових метрик плюс інтегральний показник є достатнім для коректного оцінювання ефективності методу DTP-Indie у типовому проєкті інді-розробки растрового графічного редактора. Експериментальна перевірка методу з використанням наведеної системи метрик проводиться у Розділі 4.

Сформовано дворівневу систему метрик ефективності методу DTP-Indie, що включає чотири метрики процесу управління (точність оцінки тривалості, частка повторних циклів валідації, індекс технічного боргу, частка схвалених запитів на корекцію) та чотири метрики якості продукту SUS, TFS, SLP, Flow Interruption Rate). Для кожної метрики наведено формулу обчислення, метод вимірювання та порогове значення з обґрунтуванням. Введено інтегральний показник ефективності методу E як зважену суму нормалізованих значень основних метрик, з рекомендованими ваговими коефіцієнтами для типових інді-проєктів та інтерпретацією значень. Запропонована система метрик дозволяє кількісно оцінити ефективність методу, забезпечує відтворюваність вимірювань та дозволяє порівнювати DTP-Indie з іншими методами управління в експериментальній частині дослідження.

3 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПІДТРИМКИ МЕТОДУ DTP-INDIE

3.1 Інформаційна технологія підтримки методу

Розроблений у попередньому розділі метод DTP-Indie сформульовано як концептуальну модель управління, що описує принципи, ролі та процедури роботи. Однак сам по собі метод не може бути безпосередньо застосований у реальному проєкті без відповідного інструментарію – конкретних шаблонів документів, формалізованих процедур, чек-листів та схем процесів, що перетворюють абстрактні принципи на покрокові інструкції для команди розробки. Цей підрозділ присвячений формуванню інформаційної технології підтримки методу, що включає всі необхідні артефакти для його практичного застосування інді-командою без додаткового консультування.

3.1.1 Загальна структура інформаційної технології

Інформаційна технологія підтримки методу DTP-Indie складається з чотирьох взаємопов'язаних компонентів. Перший компонент – формалізований процес управління, представлений у нотації BPMN (Business Process Model and Notation), яка є міжнародним стандартом ISO/IEC 19510 для моделювання бізнес-процесів. Другий компонент – набір шаблонів документів, які створюються та оновлюються протягом проєкту: документ бачення, план розробки ядра, беклог інструментарію та журнал запитів на корекцію. Третій компонент – формалізовані чек-листи, що використовуються для проведення гейтів валідації. Четвертий компонент – типові робочі процедури, що описують типові дії учасників команди в характерних ситуаціях проєкту.

Усі компоненти інформаційної технології спроектовані таким чином, щоб мінімізувати адміністративне навантаження на команду. Кількість документів обмежена чотирма базовими, кожен з яких має короткий шаблон з мінімально необхідним переліком полів. Чек-листи містять не більше 5–7 пунктів кожен, що дозволяє виконувати їх за час близько 15 хвилин. BPMN-схема процесу побудована як єдиний інтегрований опис, що дозволяє учаснику команди швидко орієнтуватися у поточному стані проєкту та подальших кроках.

3.1.2 Формалізований опис процесу у нотації BPMN

Процес управління за методом DTP-Indie у нотації BPMN наведено на рисунку 3.1. Діаграма побудована за стандартом BPMN 2.0 з використанням трьох плавальних доріжок (swim lanes), що відповідають трьом ролям методу – Власнику методу, Колії Ядра та Колії Інструментарію.

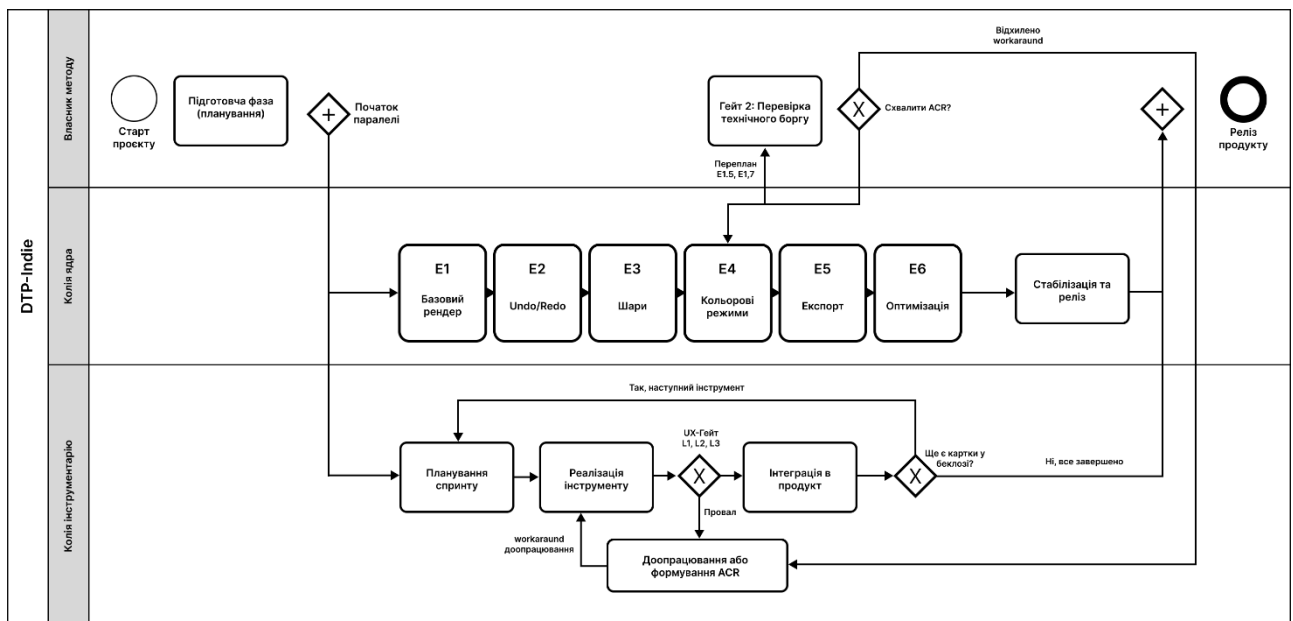


Рисунок 3.1 – BPMN-діаграма процесу управління за методом DTP-Indie

Процес починається подією старту проєкту у доріжці Власника методу. Перший крок – підготовча фаза, представлена як підпроцес (фіолетовий блок), що включає формування трьох базових артефактів проєкту. Після завершення підготовки спрацьовує паралельний шлюз (gateway зі знаком «+»), що активує одночасну роботу обох колій.

Колія Ядра представлена послідовністю шести типових етапів (E1-E6), що йдуть строго один за одним без можливості переключення. Стрілки між етапами показують жорсткий порядок виконання – це візуальне відображення предиктивної природи управління Колією Ядра.

Колія Інструментарію має циклічну структуру, що відображає її ітеративний характер. Цикл починається з планування спринту, переходить до реалізації інструмента, після чого спрацьовує гейт UX-валідації (gateway зі знаком «?»). У разі проходження гейту інструмент інтегрується у продукт, після чого паралельний шлюз запускає наступний спринт. У разі провалу інструмент потрапляє у блок «Доопрацювання або запит на корекцію», звідки можливі два шляхи: повернення на доопрацювання у Колію Інструментарію або формування запиту на корекцію ядра з ескалацією до Гейту 2 у доріжці Власника методу.

Гейт 2 (перевірка технічного боргу) має дві гілки результату: при схваленні запиту відбувається переплан плану розробки ядра і Колія Ядра тимчасово переключається на корекцію; при відхиленні запиту інструмент повертається у Колію Інструментарію з вимогою застосувати обхідне рішення. Завершальна фаза проєкту представлена підпроцесом стабілізації та релізу, що завершується кінцевою подією – публічним випуском продукту.

3.1.3 Шаблон документа бачення продукту

Документ бачення є першим артефактом, що формується на підготовчій фазі проєкту. Його призначення – зафіксувати ключові уявлення команди про продукт, його цільову аудиторію та обмеження ще до початку розробки. Документ повинен залишатися коротким (1–2 сторінки) та оновлюватися рідко – лише у випадках принципової зміни напрямку проєкту. Структура шаблону документа наведена у Додатку Б.

Документ бачення створюється у форматі простого текстового файлу і зберігається у спільному робочому просторі команди (наприклад, у Notion або Google Docs). Він є точкою референсу для всіх подальших рішень: при виникненні суперечок щодо пріоритетів команда повертається до документа бачення для звірки із спочатку зафіксованою цінністю продукту.

3.1.4 Шаблон плану розробки ядра

План розробки ядра (Engine Roadmap) є центральним артефактом Колії Ядра. На відміну від документа бачення, що описує продукт як ціле, план фокусується виключно на архітектурі графічного движка та послідовності його реалізації. Структура шаблону наведена у Додатку В.

Ключова особливість плану розробки ядра – наявність окремого журналу перепланів, у якому фіксуються всі затверджені зміни до початкової послідовності етапів. Цей журнал дозволяє ретроспективно проаналізувати, які архітектурні припущення виявилися хибними, і використати ці знання для калібрування планів у майбутніх проєктах.

3.1.5 Шаблон беклогу інструментарію та журналу запитів на корекцію

Беклог інструментарію та журнал запитів на корекцію (ACR-журнал) – це два логічно різних, але технічно об'єднаних документи, що зберігаються у вигляді таблиці у спільному просторі команди. Вони оновлюються найбільш активно з усіх артефактів і є основним інструментом оперативного управління Колією Інструментарію, шаблони представлено у Додатку Г та Додатку Д.

Інтегрований підхід до зберігання беклогу та журналу ACR має важливу методологічну цінність: він дозволяє швидко відстежити причинно-наслідкові зв'язки між появою інструмента у беклозі і виникненням пов'язаних з ним архітектурних змін. Це є основою для аналізу метрики (частка схвалених запитів на корекцію), описаної у підрозділі 2.4.2.

3.1.6 Чек-листи гейтів валідації

Гейти валідації є ключовими процедурами методу DTP-Indie, що визначають моменти прийняття управлінських рішень. Для забезпечення відтворюваності цих процедур різними учасниками команди – і для уникнення суб'єктивності – гейти представлені у вигляді формалізованих чек-листів. Кожен чек-лист є окремим документом, що використовується щоразу при проходженні відповідного гейту.

Чек-лист Рівня 1 гейту UX-валідації – експрес-перевірка ергономіки інструмента, що виконується Власником методу безпосередньо після завершення розробки інструмента.

Чек-лист Рівня 1 – експрес-перевірка ергономіки:

- доступ до основної функції інструмента – не більше трьох кліків від запуску застосунку;
- іконка інструмента впізнавана без супровідного підпису (підтверджено мінімум 3 з 5 членами команди);
- поведінка інструмента передбачувана – немає прихованих побічних ефектів;
- зворотність дії через стандартну операцію Ctrl+Z працює коректно;
- відсутні обов'язкові модальні діалоги для базових операцій;
- інструмент має зрозумілий стан «активний / неактивний»;
- реакція інструмента на дію користувача миттєва (без видимої затримки);

Чек-лист Рівня 2 гейту UX-валідації – кількісне тестування з зовнішніми тестерами. Виконується протягом 1–2 годин з 3–5 учасниками.

Чек-лист Рівня 2 – кількісне UX-тестування:

- підготовлено тестове завдання (типовий сценарій використання інструмента);
- залучено мінімум 3 тестери, що раніше не працювали з продуктом;
- виміряно час до першого мазка (TFS) для кожного тестера, обчислено медіану;
- виміряно затримку реакції системи (SLP) інструментально або суб'єктивно;
- зафіксовано кількість переривань творчого процесу (FIR) за весь сеанс;
- заповнено опитування SUS (10 стандартних запитань) кожним тестером;
- усі чотири метрики відповідають пороговим значенням (див. підрозділ 2.4).

Чек-лист Рівня 3 гейту UX-валідації – перевірка архітектурної сумісності, виконується Розробником ядра.

Чек-лист Рівня 3 – архітектурна сумісність:

- інструмент не потребує змін у затвердженому публічному API ядра;

- у реалізації інструмента відсутні обхідні шляхи, що компенсують відсутність потрібного функціоналу в ядрі;
- інструмент не створює нових жорстких залежностей між модулями;
- інструмент покривається автоматизованими тестами на критичні сценарії;
- якщо хоча б один пункт не виконано – формується запит на корекцію ядра.

Окремий чек-лист застосовується для проведення Гейту 2 – перевірки технічного боргу при отриманні запиту на корекцію ядра. Виконується спільно Розробником ядра та Власником методу.

Чек-лист Гейту 2 – перевірка технічного боргу:

- чи дійсно неможливо реалізувати інструмент через обхідне рішення без зміни ядра?
- Чи виправдана зміна співвідношенням вартості робіт до отриманої функціональності?
- Чи має зміна системний вплив на майбутні інструменти, або стосується лише поточного?
- Чи зберігається стабільність публічного API ядра після впровадження змін?
- Чи не порушують зміни загальну логіку плану розробки ядра?
- Прийнято рішення: Схвалити/Відхилити/Повернути на доопрацювання

3.1.7 Типові робочі процедури

Окрім формалізованих документів та чек-листів, інформаційна технологія DTP-Indie включає опис типових робочих процедур – послідовностей дій

учасників команди в характерних ситуаціях проєкту. Розглянемо три найважливіші процедури.

Процедура 1: Старт нового спринту інструментарію. Виконується Розробником інструментарію спільно з Власником методу один раз на 1–2 тижні. Послідовність кроків:

- 1) переглянути беклог інструментарію та обрати 1–2 інструменти найвищого пріоритету, реалізація яких підтримується поточним станом ядра;
- 2) перевірити, що обрані інструменти не потребують змін у ядрі, які зараз заплановані як майбутні етапи плану ядра;
- 3) сформулювати критерії готовності для обраних інструментів ;
- 4) зафіксувати спринт у беклозі зі статусом «У спринті». Тривалість процедури – 15–30 хвилин.

Процедура 2: Проходження гейту UX-валідації. Виконується одразу після завершення розробки інструмента. Послідовність кроків:

- 1) Розробник інструментарію передає інструмент на тестування Власнику методу;
- 2) Власник методу проходить чек-лист Рівня 1 за 15 хвилин;
- 3) у разі успіху організовується сесія тестування Рівня 2 з зовнішніми тестерами (1–2 години);
- 4) Розробник ядра проходить чек-лист Рівня 3 за 30 хвилин;
- 5) у разі провалу будь-якого рівня заповнюється запис у беклозі зі статусом «На доопрацюванні»;
- 6) при провалі Рівня 3 формується запит на корекцію ядра з ініціюванням Гейту 2.

Процедура 3: Обробка запиту на корекцію ядра. Виконується при провалі інструмента на Рівні 3 гейту UX-валідації. Послідовність кроків:

- 1) Розробник інструментарію заповнює шаблон запиту у журналі ACR (10–15 хвилин);

- 2) Власник методу та Розробник ядра проводять архітектурний review за чек-листом Гейту 2 (30–60 хвилин);
- 3) рішення фіксується у журналі ACR з обґрунтуванням;
- 4) при схваленні Розробник ядра планує під-етап у плані розробки ядра і тимчасово переключається на корекцію;
- 5) при відхиленні Розробник інструментарію повертає інструмент у беклог з позначкою «потребує обхідний шлях» та плануванням обхідного рішення;
- 6) Інструмент повертається на повторну UX-валідацію після завершення корекції або реалізації обхідного шляху.

Описані процедури повинні стати фактично рутиною для команди – настільки, що члени команди виконують їх автоматично, без необхідності щоразу звертатися до документації. Саме така рутинізація процедур забезпечує перетворення методу з абстрактної концепції на реальну управлінську практику.

3.1.8 Інтеграція компонентів інформаційної технології

Усі компоненти описаної інформаційної технології – BPMN-схема процесу, шаблони документів, чек-листи та робочі процедури – пов'язані між собою у єдину систему. BPMN-схема визначає загальну послідовність дій; шаблони документів забезпечують структуру для фіксації результатів цих дій; чек-листи формалізують прийняття рішень у ключових точках процесу; робочі процедури описують типові послідовності виконання чек-листів та оновлення документів.

Така інтегрованість дозволяє команді працювати за методом DTP-Indie без необхідності в окремому навчанні чи зовнішньому консультуванні. Достатньо отримати комплект артефактів інформаційної технології, ознайомитися з BPMN-

схемою процесу і розпочати виконання робочих процедур, заповнюючи шаблони документів за встановленими формами.

Сформовано інформаційну технологію підтримки методу DTP-Indie, що включає чотири взаємопов'язані компоненти: формалізований опис процесу у нотації BPMN з трьома плавальними доріжками для ролей методу; набір шаблонів чотирьох базових документів (документ бачення, план розробки ядра, беклог інструментарію, журнал запитів на корекцію); чотири чек-листи для гейтів валідації (три рівні UX-валідації та перевірка технічного боргу); три типові робочі процедури, що покривають основні управлінські ситуації методу. Усі компоненти інформаційної технології спроектовані за принципом ресурсної ефективності – мінімальної кількості документів та формальних процедур, достатньої для коректної реалізації методу інді-командою. Розгляд конкретного інструментарію для імплементації інформаційної технології у робочих процесах команди проводиться у наступному підрозділі.

3.2 Інструментарій імплементації інформаційної технології

Розроблена у попередньому підрозділі інформаційна технологія підтримки методу DTP-Indie описує процес, документи та процедури на концептуальному рівні. Для практичного застосування методу в реальних робочих процесах інді-команди ці абстрактні артефакти повинні бути реалізовані у конкретних програмних інструментах, доступних для команди розробників. Цей підрозділ розглядає вибір інструментарію імплементації, обґрунтовує переваги стеку Notion і Trello для умов інді-розробки, демонструє конкретну конфігурацію цих інструментів під метод DTP-Indie на прикладі гіпотетичного проєкту PixelLab та описує типову модель роботи менеджера команди у щоденному режимі.

3.2.1 Огляд інструментарію для інді-команд

Сучасний ринок інструментів управління ІТ-проектами пропонує широкий спектр рішень, які умовно поділяються на три категорії за рівнем складності та цільовим масштабом використання.

Перша категорія – корпоративні системи управління (Jira, Azure DevOps, IBM Rational). Ці інструменти орієнтовані на великі команди розробки з виокремленими ролями менеджера проєкту, скрам-майстра, аналітиків. Вони пропонують глибокі можливості налаштування процесів, складні системи звітності, інтеграцію з корпоративними системами безпеки та аудиту. Водночас їх використання вимагає значних адміністративних витрат на налаштування та підтримку, а вартість ліцензій робить їх економічно недоцільними для команд з 2–7 осіб. Для методу DTP-Indie, цільовою аудиторією якого є інді-команди, корпоративні системи не розглядаються як основний варіант імплементації.

Друга категорія – спеціалізовані інструменти гнучкого моделювання (Linear, ClickUp, Monday.com). Ці продукти орієнтовані на середні команди і пропонують компроміс між функціональною повнотою корпоративних систем та простотою використання. Вони мають вбудовані механізми спринтів, беклогу, канбан-дошок. Проте більшість таких інструментів є платними з підпискою від 8–15 доларів на місяць за користувача, що для інді-команд з 2–3 осіб становить значні витрати у довгостроковій перспективі. Крім того, ці продукти продовжують бути комплексними у використанні, що збільшує час адаптації нових учасників команди.

Третя категорія – універсальні інструменти продуктивності (Notion, Trello, Asana). Ці платформи не позиціонуються виключно як інструменти управління ІТ-проектами, але завдяки гнучкості налаштування дозволяють реалізувати практично будь-який методологічний підхід. Вони пропонують безкоштовні плани, достатні для команд до 10 осіб, інтуїтивні інтерфейси з мінімальним

порогом входження та широку екосистему інтеграцій. Саме ця категорія найбільш відповідає вимогам інді-розробки, де ресурсна ефективність є критичним фактором.

На основі проведеного аналізу для імплементації методу DTP-Indie у цьому дослідженні обрано стек з двох інструментів – Notion і Trello, що працюють у комплементарній ролі. Цей вибір обґрунтовується чотирма ключовими факторами. По-перше, обидва інструменти є безкоштовними у базових планах, що відповідає принципу ресурсної ефективності методу. По-друге, вони мають мінімальний поріг входження, що дозволяє інді-команді почати використання без спеціального навчання. По-третє, ця пара інструментів забезпечує природне функціональне розділення: Notion слугує сховищем документації та структурованих даних проєкту, тоді як Trello підтримує оперативне управління поточними роботами Колії Інструментарію. По-четверте, обидва інструменти мають кросплатформну підтримку (Windows, macOS, веб, мобільні платформи), що дозволяє членам команди працювати у будь-якому середовищі.

Запропоноване функціональне розділення інструментарію між Notion і Trello відповідає природному розмежуванню типів артефактів методу DTP-Indie. Notion відповідає за документи з структурованими даними, що оновлюються рідко та потребують детального ведення – план розробки ядра, журнал запитів на корекцію, документ бачення продукту, чек-листи валідації. Trello відповідає за оперативне відображення стану Колії Інструментарію – потік карток-інструментів через спринти, гейти валідації та інтеграцію у продукт. Така спеціалізація дозволяє кожному інструменту реалізовувати свою сильну сторону без надлишкової складності.

3.2.2 Конфігурація робочого простору Notion

Робочий простір Notion для проєкту, що ведеться за методом DTP-Indie, організовується у вигляді ієрархічної структури з єдиною кореневою сторінкою – Project Hub. Кореневу сторінку формує заголовок проєкту, короткий опис команди та методу управління, а також список вкладених сторінок, що відповідають чотирьом базовим артефактам методу. На рисунку 3.2 наведено приклад конфігурації Project Hub для гіпотетичного проєкту PixelLab.

Як видно з рисунка 3.2, Project Hub виконує функцію точки входу до всіх артефактів проєкту. Користувач, відкривши робочий простір, одразу отримує загальне уявлення про команду, метод управління та структуру документації. Подальша робота з конкретними артефактами здійснюється через перехід на вкладені сторінки.

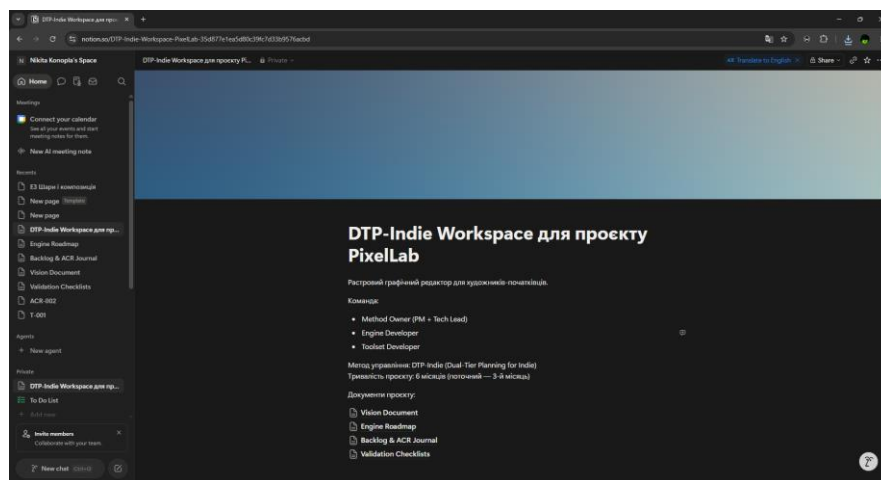


Рисунок 3.2 – Notion головна сторінка Project Hub, де видно всі 4 документи

Vision Document (рисунки 3.3 та 3.4) реалізується як проста текстова сторінка з ієрархічною структурою заголовків. Документ містить сім розділів: назва продукту, цільова аудиторія, ключова цінність, базовий функціонал,

технологічний стек, обмеження проєкту, UX-пороги для валідації. Сторінка не призначена для постійного оновлення – лише у випадках принципової зміни напряму проєкту. Текстова організація без баз даних відповідає природі документа: його зміст не потребує сортування або фільтрації.

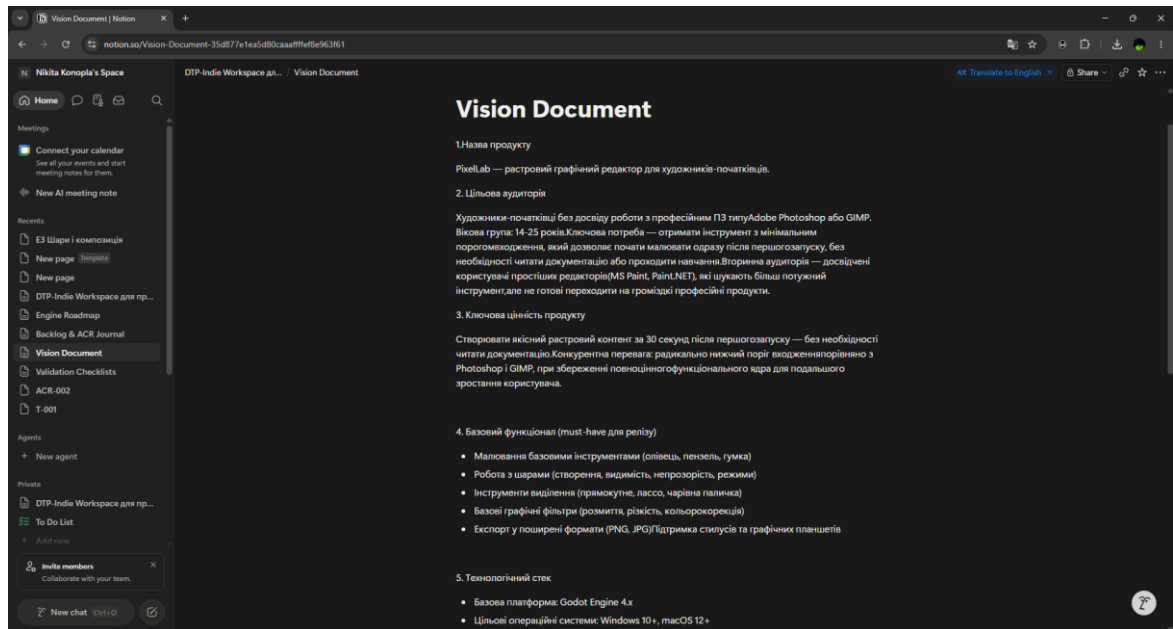


Рисунок 3.3 — Notion сторінка Vision Document

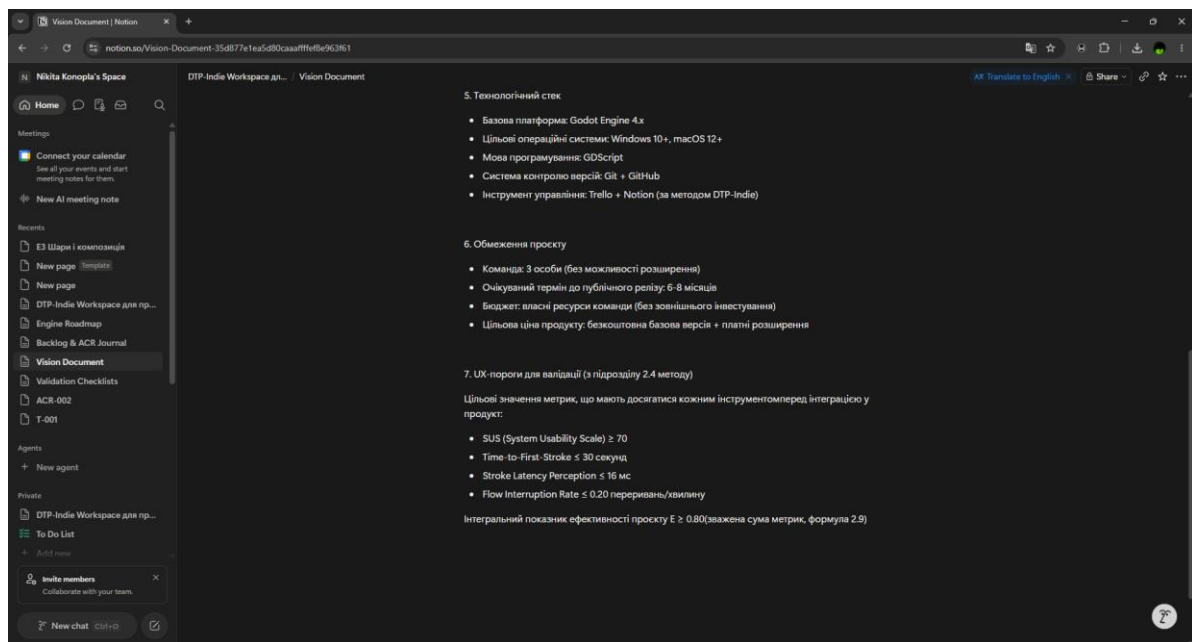
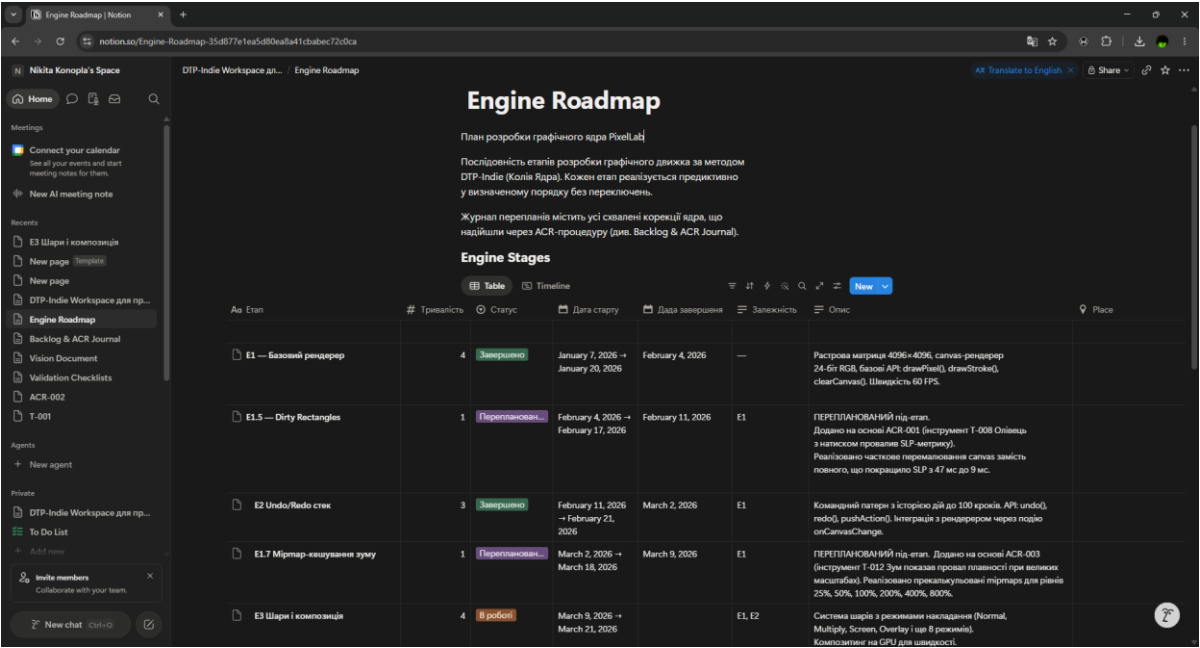


Рисунок 3.4 — Notion сторінка Vision Document (кінець)

Engine Roadmap реалізується як база даних типу Table view зі спеціалізованими властивостями для управління планом розробки ядра. Кожен рядок таблиці відповідає одному етапу плану і містить сім полів: назва етапу, тривалість у тижнях, статус (Завершено/В роботі/Заплановано/Перепланований), дата старту, дата завершення, залежності між етапами та текстовий опис. Під таблицею розташовано окремий блок «Журнал перепланів», що документує всі схвалені корекції плану з посиланнями на відповідні запити на корекцію ядра.

На рисунках 3.5, 3.6 та 3.7 наведено приклад заповненого Engine Roadmap для проєкту PixelLab на 14-му тижні розробки. У таблиці видно два завершених основних етапи (E1 та E2), два перепланованих під-етапи (E1.5 та E1.7, що з'явилися внаслідок схвалених запитів на корекцію), один поточний етап (E3) та три заплановані етапи (E4, E5, E6).



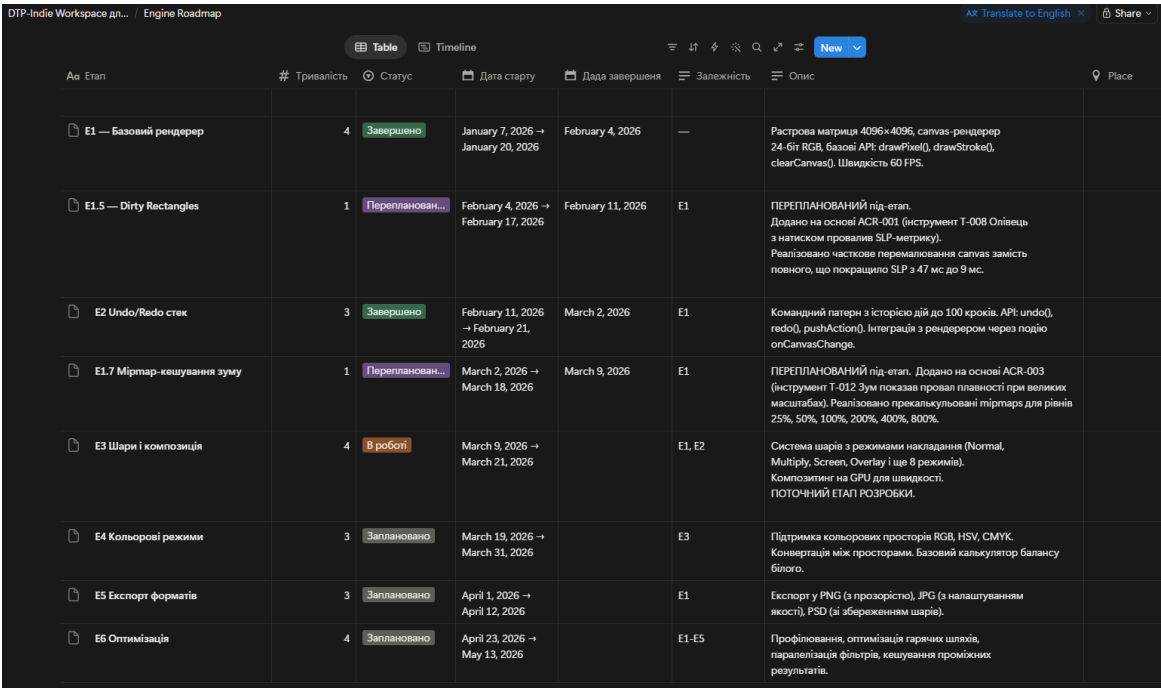
Engine Roadmap						
План розробки графічного ядра PixelLab						
Послідовність етапів розробки графічного движка за методом DTP-Indie (Колія Ядра). Кожен етап реалізується предиктивно у визначеному порядку без переключень.						
Журнал перепланів містить усі схвалені корекції ядра, що надійшли через ACR-процедуру (див. Backlog & ACR Journal).						
Engine Stages						
№	Етап	Тривалість	Статус	Дата старту	Дата завершення	Залежність
1	E1 — базовий рендерер	4	Завершено	January 7, 2026 → January 20, 2026	February 4, 2026	—
2	E1.5 — Dirty Rectangles	1	Переплановано	February 4, 2026 → February 17, 2026	February 11, 2026	E1
3	E2 Undo/Redo стек	3	Завершено	February 11, 2026 → February 21, 2026	March 2, 2026	E1
4	E1.7 Міртам-включення звуку	1	Переплановано	March 2, 2026 → March 18, 2026	March 9, 2026	E1
5	E3 Шари і композиція	4	В роботі	March 9, 2026 → March 21, 2026		E1, E2

Рисунок 3.5 – Notion Engine Roadmap, повна таблиця Engine Stages

Структура нумерації під-етапів (E1.5, E1.7) реалізує принцип розширення плану без переномерування існуючих етапів. Кожен під-етап залишається

семантично прив'язаним до свого базового етапу (E1.x належить до групи E1), що зберігає референційну цілісність посилань у беклозі інструментарію та журналі запитів на корекцію. Така схема дозволяє додавати нові архітектурні корекції без порушення вже існуючих залежностей.

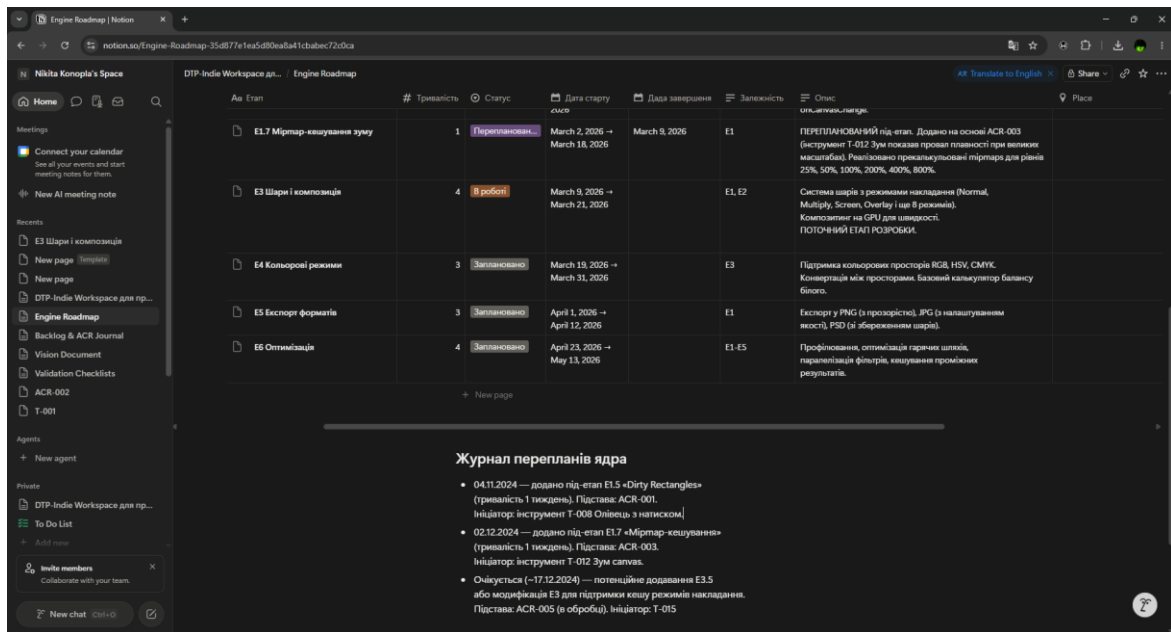
Окрім табличного представлення (Table view), Engine Roadmap у Notion підтримує альтернативне представлення тих самих даних – Timeline view, що візуалізує етапи плану на часовій шкалі.



Ал	Етап	#	Тривалість	Статус	Дата старту	Дата завершення	Залежність	Опис	Place
	E1 — Базовий рендерер	4	Завершено	January 7, 2026 → January 20, 2026	February 4, 2026	—		Растрова матриця 4096x4096, canvas-рендерер 24-bit RGB, базові API: drawPixel(), drawStroke(), clearCanvas(). Швидкість 60 FPS.	
	E1.5 — Dirty Rectangles	1	Переплановано	February 4, 2026 → February 17, 2026	February 11, 2026	E1		ПЕРЕПЛАНОВАНИЙ під-етап. Додано на основі ACR-001 (інструмент T-008 Олівець з натиском провалив SLP-метрику). Реалізовано часткове перемалювання canvas замість повного, що покращило SLP з 47 мс до 9 мс.	
	E2 Undo/Redo стек	3	Завершено	February 11, 2026 → February 21, 2026	March 2, 2026	E1		Командний ланцюг з історією дій до 100 кроків. API: undo(), redo(), pushAction(). Інтеграція з рендерером через подію onCanvasChange.	
	E1.7 Мірпар-кешування зуму	1	Переплановано	March 2, 2026 → March 18, 2026	March 9, 2026	E1		ПЕРЕПЛАНОВАНИЙ під-етап. Додано на основі ACR-003 (інструмент T-012 Зум показав провал плавності при великих масштабах). Реалізовано прекалькульовані мірпары для рівнів 25%, 50%, 100%, 200%, 400%, 800%.	
	E3 Шари і композиція	4	В роботі	March 9, 2026 → March 21, 2026		E1, E2		Система шарів з режимами накладання (Normal, Multiply, Screen, Overlay і ще 8 режимів). Композитинг на GPU для швидкості. ПОТОЧНИЙ ЕТАП РОЗРОБКИ.	
	E4 Кольорові режими	3	Заплановано	March 19, 2026 → March 31, 2026		E3		Підтримка кольорових просторів RGB, HSV, CMYK. Конвертація між просторами. Базовий калькулятор балансу білого.	
	E5 Експорт форматів	3	Заплановано	April 1, 2026 → April 12, 2026		E1		Експорт у PNG (з прозорістю), JPG (з налаштуванням якості), PSD (зі збереженням шарів).	
	E6 Оптимізація	4	Заплановано	April 23, 2026 → May 13, 2026		E1-E5		Профілювання, оптимізація гарячих шляхів, паралелізація фільтрів, кешування проміжних результатів.	

Рисунок 3.6 – Notion Engine Roadmap, повна таблиця Engine Stages (продовження)

Timeline view є по суті спрощеною формою діаграми Ганта, адаптованою під потреби інді-методу. Кожен етап представлений горизонтальною смугою, довжина якої пропорційна тривалості. Така візуалізація дозволяє команді отримати миттєве цілісне бачення прогресу проєкту без необхідності переходу до табличних деталей. На рисунку 3.8 наведено Timeline view проєкту PixelLab у масштабі Year, що охоплює весь шестимісячний цикл розробки.



Назва	Тривалість	Статус	Дата початку	Дата завершення	Залежності	Опис
E1.7 Мірпак-кешування зуму	1	Перепланований	March 2, 2026 → March 18, 2026	March 9, 2026	E1	ПЕРЕПЛАНОВАНИЙ під-етап. Додано на основі ACR-003 (інструмент T-012 Зум показав провал плавності при великих масштабах). Реалізовано пререндашоновані мірпаки для рівнів 25%, 50%, 100%, 200%, 400%, 800%.
E3 Шари і композиція	4	В роботі	March 9, 2026 → March 21, 2026		E1, E2	Система шарів з режимами накладання (Normal, Multiply, Screen, Overlay і ще 6 режимів). Конверсія на GPU для швидкості. ПОТОЧНИЙ ЕТАП РОЗРОБКИ.
E4 Колірний режим	3	Запланований	March 19, 2026 → March 31, 2026		E3	Підтримка кольорових просторів RGB, HSV, CMYK. Конверсія між просторами, базовий калькулятор балансу білого.
E5 Експорт формату	3	Запланований	April 1, 2026 → April 12, 2026		E1	Експорт у PNG (з прозорістю), JPG (з налаштуванням якості), PSD (з збереженням шарів).
E6 Оптимізація	4	Запланований	April 23, 2026 → May 13, 2026		E1-E5	Профілювання, оптимізація гарячих шляхів, паралелізація фільтрів, кешування проміжних результатів.

Журнал перепланів ядра

- 04.11.2024 — додано під-етап E1.5 «Dirty Rectangles» (тривалість 1 тиждень). Підстава: ACR-001. Ініціатор: інструмент T-008 Олівець з натиском.
- 02.12.2024 — додано під-етап E1.7 «Мірпак-кешування» (тривалість 1 тиждень). Підстава: ACR-003. Ініціатор: інструмент T-012 Зум canvas.
- Очікується (~17.12.2024) — потенційне додавання E3.5 або модифікація E3 для підтримки кешу режимів накладання. Підстава: ACR-005 (в обробці). Ініціатор: T-015.

Рисунок 3.7 – Notion Engine Roadmap, таблиця Engine Stages (кінець)

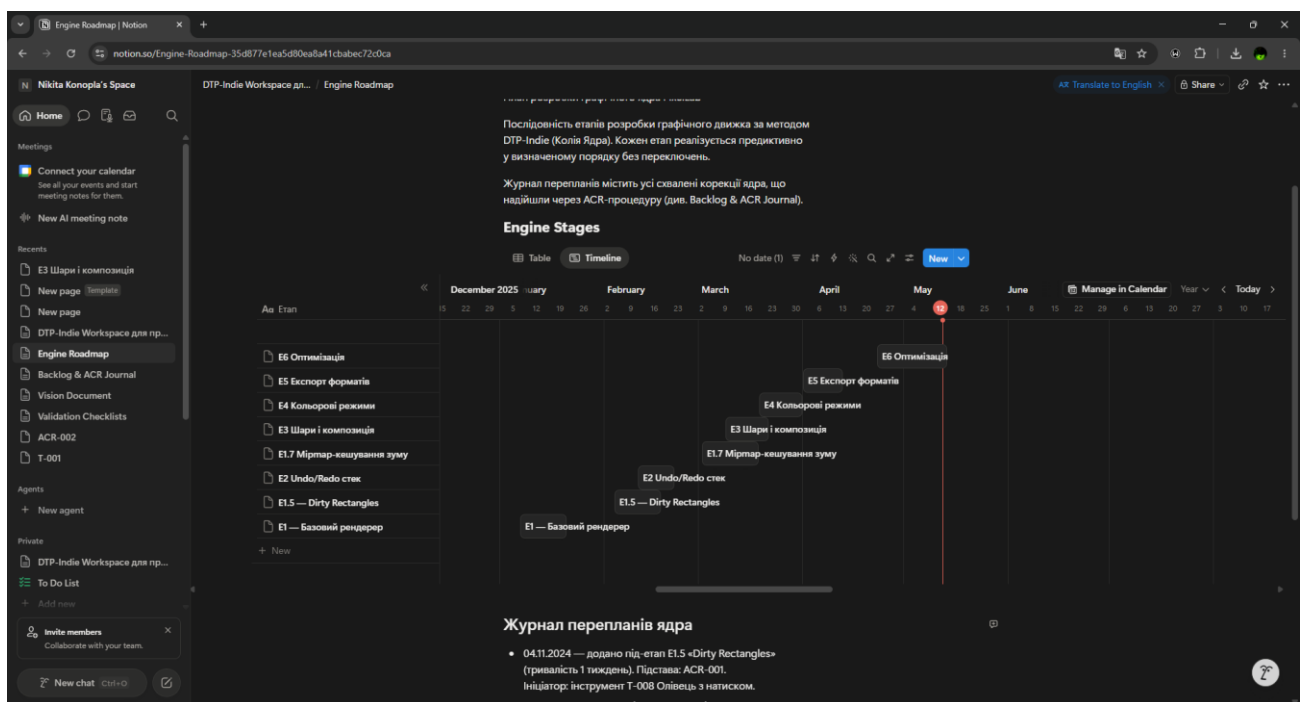


Рисунок 3.8 – Notion Timeline view бази Engine Stages у масштабі Year, на якому видно всі 8 етапів у вигляді горизонтальних смуг

Принципово важливо, що Timeline view відображає лише Колію Ядра – Колія Інструментарію має іншу темпоральну логіку (короткі спринти 1–2 тижні без жорсткої прив'язки до календарних дат), що робить її непридатною для представлення у форматі Ганти. Це є прямим наслідком ключового принципу методу DTP-Indie – двоколієвого підходу з різними парадигмами управління. Класична монолінійна діаграма Ганта, що традиційно застосовується у РМВОК-орієнтованих методах для всього проєкту, у методі DTP-Indie замінена на двокомпонентну структуру: Timeline view Engine Roadmap для предиктивно керованої колії та беклог зі складністю у спринтах для адаптивної колії. Інтегральне візуальне представлення обох колій разом наводиться у Розділі 4 як частина експериментальної перевірки методу [16].

Перемикання між Table view і Timeline view у Notion здійснюється одним кліком за допомогою вкладок над базою даних. Обидва представлення працюють з тими ж самими даними – внесення змін у будь-якому view одразу відображається в іншому. Така інтегрованість дозволяє різним членам команди використовувати найбільш зручне для їх задач представлення: Розробник ядра працює з табличним видом для введення дат і статусів, тоді як Власник методу використовує часову шкалу для контролю загального прогресу.

Backlog & ACR Journal є центральним документом оперативного управління проєктом і реалізується як сторінка з двома послідовними базами даних. Перша база – Toolset Backlog – містить повний перелік запланованих інструментів продукту з повним набором характеристик. Друга база – ACR Journal – фіксує всі запити на корекцію ядра, що подавалися протягом проєкту, з рішеннями та рефлексіями для подальшого калібрування.

На рисунку 3.9 наведено фрагмент Toolset Backlog для проєкту PixelLab.

Backlog & ACR Journal

Беклог інструментарію та журнал запитів на корекцію ядра.

Беклог містить повний перелік запланованих інструментів продукту. ACR Journal фіксує всі архітектурні запити, що подавалися протягом спринту, з рішеннями та рефлексіями для подальшого калібрування.

Toolset Backlog

ID	Назва	Пріоритет	Залежність	# Складність	Статус	# SUS	# TFS	# SLP	# FIR
T-001	Опівід базовий	High	E1	0.5	Integrated	82	18	12	0.08
T-002	Пензель круглий	High	E1	1	Integrated	85	22	14	0.1
T-003	Гумка	Medium	E1	0.5	Integrated	80	16	11	0.05
T-004	Заливка	Medium	E1	1	Integrated	75	28	15	0.12
T-005	Виділення прямокутне	High	E2	1	Integrated	77	25	13	0.14
T-006	Виділення лассо	High	E2	1	Integrated	72	30	14	0.16
T-007	Виділення лассо	Medium	E2	1	Integrated	71	32	13	0.18
T-008	Опівід з написом	Medium	E1.5	1.5	Integrated	86	20	9	0.06
T-009	Пензель з прозорістю	High	E1	0.5	Integrated	81	24	12	0.09
T-010	Заливка розміру canvas	High	E1	1	Integrated	85	22	12	0.08
T-011	Поворот canvas	Low	E1	0.5	Integrated	80	16	14	0.1
T-012	Зум canvas (повільний)	High	E1.7	1	Integrated	75	28	11	0.05
T-013	Шарик створити/видалити	High	E1	1	Integrated	77	25	15	0.12
T-014	Шарик виділити і непрозорість	High	E1	1	Integrated	72	30	13	0.14
T-015	Шарик розміри налаштування	High	E1	1	Gate 1	71	32	14	0.16
T-016	Інструмент калібрування	Medium	E2	1.5	Gate 1	80	20	15	0.18

Рисунок 3.9 – Notion Toolset Backlog, та журнал запитів ACR на зміну ядра

Таблиця містить тридцять два інструменти, що розподілені за статусами життєвого циклу: чотирнадцять інтегрованих у продукт (з повним набором метрик SUS, TFS, SLP та FIR за результатами проходження гейту UX-валідації), три на різних рівнях гейту валідації, два у поточному спринті розробки, три заплановані у наступний спринт та десять у беклозі. Така структура дозволяє командам у будь-який момент отримати повну картину обсягу робіт за Колією Інструментарію.

На рисунку 3.10 наведено ACR Journal проєкту PixelLab з повною історією запитів на корекцію ядра на 14-му тижні розробки. Журнал містить сім записів, що відповідають усім архітектурним корекціям, які подавалися протягом проєкту.

Аналіз журналу дозволяє виявити закономірності в роботі команди з архітектурою ядра. З семи запитів чотири схвалено (ACR-001, ACR-003, ACR-004, з очікуваним схваленням ACR-005), що призвело до відповідного перепланування плану розробки ядра. Два запити відхилено (ACR-002 та ACR-

006): команда визначила, що проблеми можна вирішити через обхідний шлях у Колії Інструментарію без зміни ядра. Один запит (ACR-007) щойно подано і очікує розгляду на Гейті 2.

ACR ID	Дата подані	Ініціатор	Проблема	Рішення	Статус
ACR-001	January 17, 2026	T-008	Інструмент T-008 показав затримку 47 мс між рухом стилуса і появою штриха при швидкому малюванні. SLP перевищує поріг 16 мс утрим.	Dirty rectangles рендеринг замість повного пере малювання canvas. Workaround неможливий — це системна проблема рендерера.	Сквалено
ACR-002	January 25, 2026	T-017	Інструмент T-017 у початковій версії потребував складного зберігання тексту в реальному часі під час редагування. Архітектура ядра не передбачала текстового буфера.	Винесення текстового буфера в окремий модуль ядра. Workaround через render-to-texture.	Відхилено
ACR-003	February 2, 2026	T-012	Зум canvas. Інструмент T-012 при масштабуванні до 400% і вище показав фрагментарну повільність зуму — затримка 38 мс на крок зміни масштабу.	Мітмар-кешування для рівнів зуму. Можлива через CPU-обчислення, але провал SLP-порогу.	Сквалено
ACR-004	February 10, 2026	T-018	Піпетка кольору. T-018 при масштабуванні canvas повертав збіжні координати пікселя — піпетка зчитувала колір з неправильної позиції.	Зчитування пікселя з урахуванням поточного масштабу. Складний workaround з обчисленням координат на льоту.	Сквалено
ACR-005	March 13, 2026	T-015	Шарік накладання. Інструмент T-015 при перемиканні режиму накладання шару викликає повільний перепаунок всіх шарів. SLP виходить за межі (24 мс при порозі 16 мс).	Кеш режимів накладання при перемиканні режиму шару. Workaround через перепаунок всіх шарів (провалює SLP).	В обробці
ACR-006	March 15, 2026	T-020	Фільтр розмиття Гауса. Фільтр Гауса (T-020) блокує UI на 2-3 секунди при застосуванні до canvas (FPS ~ 0.026). Координати не	Окремий потік для виконання фільтрів. Виконання у поточному UI-потязі з прогрес-баром.	Відхилено

Рисунок 3.10 – Notion ACR Journal

Окремої уваги заслуговують поля «Обсяг очікуваний» та «Обсяг» — це попередня оцінка та фактичні витрати часу на реалізацію корекції. Зіставлення цих двох значень дозволяє обчислювати метрику ЕА, описану у підрозділі 2.4. Для проєкту PixelLab на 14-му тижні точність оцінки складає в середньому близько 90 відсотків — приклад показує одну незначну недооцінку на 1 день у ACR-003.

Колонка «Проблема» документує конкретну ситуацію, що спричинила запит, з кількісними характеристиками (наприклад, «затримка 47 мілісекунд при швидкому малюванні» для ACR-001). Ця інформація використовується як обґрунтування рішення Гейту 2 та забезпечує можливість ретроспективного аналізу патернів виникнення архітектурних проблем у проєкті.

Validation Checklists (рисунки 3.11 та 3.12) реалізуються як сторінка з чотирма toggle-блоками, що відповідають чотирьом ключовим процедурам валідації методу. Toggle-структура дозволяє тримати всі чек-листи на одній сторінці у згорнутому стані, що покращує навігацію та зменшує когнітивне навантаження на учасника команди при щоденному використанні.

Чотири чек-листи відповідають трьом рівням гейту UX-валідації (Рівень 1 – експрес-перевірка ергономіки, Рівень 2 – кількісне UX-тестування, Рівень 3 – архітектурна сумісність) та Гейту 2 (перевірка технічного боргу). Notion-версія чек-листів виконує роль майстер-шаблону: усі пункти зберігаються тут у пустому стані. Фактичне проходження валідації для конкретного інструмента документується безпосередньо у Trello на картці відповідного інструмента, де чек-лист копіюється з Notion і заповнюється під час реальної процедури.

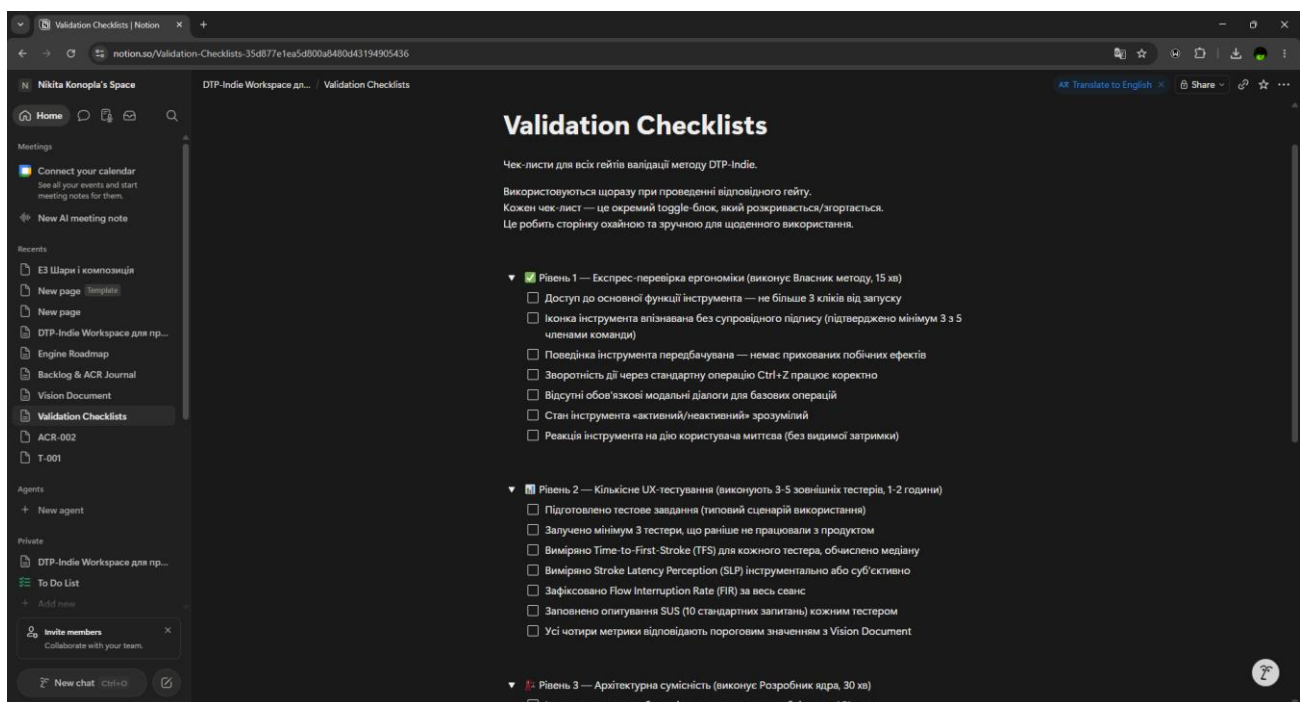


Рисунок 3.11 – Notion Validation Checklists

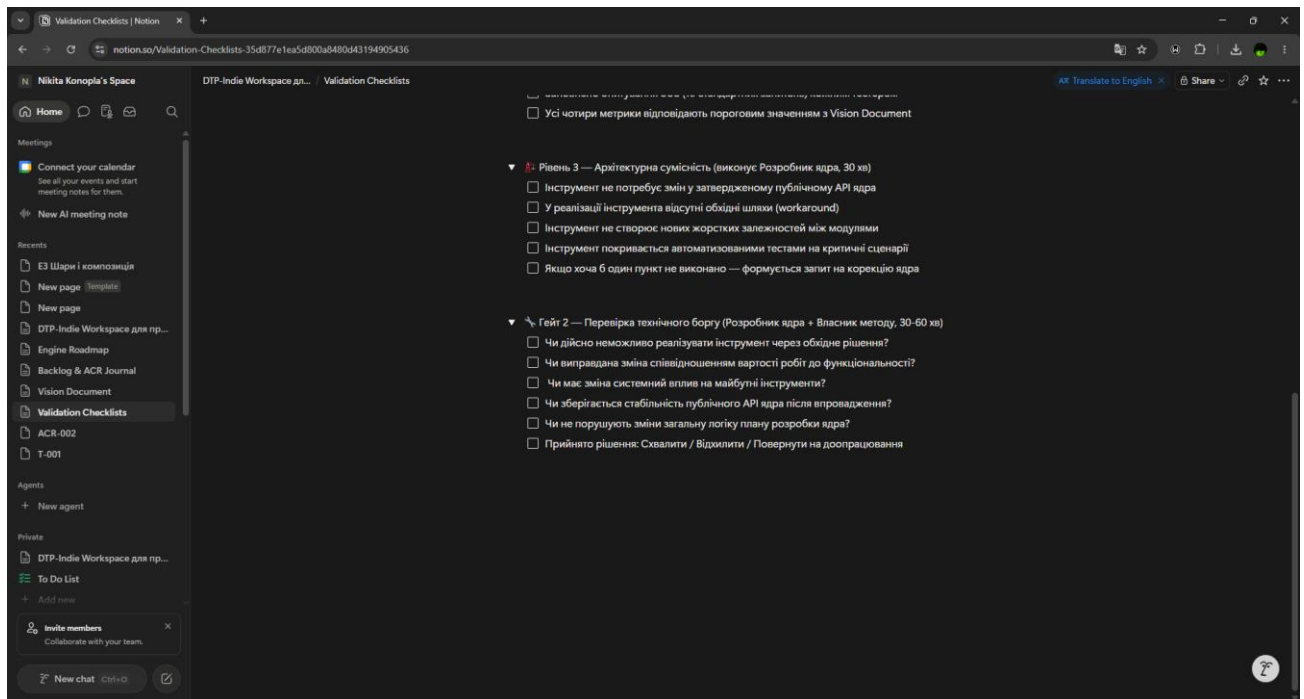


Рисунок 3.12 – Notion Validation Checklists (кінець)

3.2.3 Конфігурація дошки Trello

Дошка Trello реалізує оперативний рівень управління Колією Інструментарію – щоденне переміщення інструментів між стадіями життєвого циклу від беклогу до інтеграції у продукт. На відміну від Notion, що зберігає структуровану документацію та історичні дані проєкту, Trello відображає лише поточний стан робіт у наочній kanban-формі.

Структура дошки (рисунок 3.13) відповідає природній послідовності проходження інструмента через метод DTP-Indie і складається з семи колонок (списків): Backlog (загальний резерв інструментів), Sprint (інструменти, обрані для поточного спринту), In Progress (інструменти в активній розробці), три колонки гейту валідації – Gate L1, Gate L2, Gate L3 – та Integrated (інструменти, інтегровані у продукт). Карта-інструмент послідовно проходить ці колонки зліва направо, відображаючи свій реальний стан у методі.

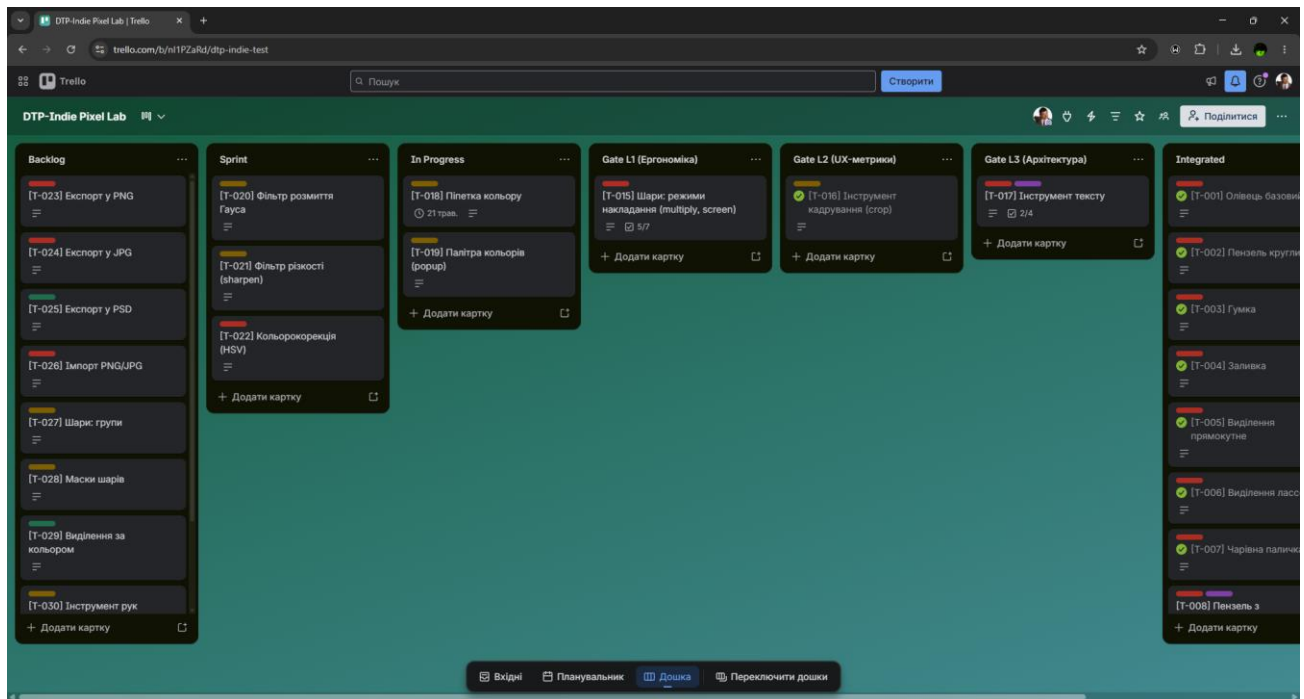


Рисунок 3.13 – Trello повна дошка проєкту. Toolset Track

На рисунку 3.13 наведено повний стан дошки проєкту PixelLab на 14-му тижні розробки. Розподіл карток по колонках відображає типову ситуацію індивідуальної команди на середині життєвого циклу проєкту: значна частина роботи вже виконана (чотирнадцять інструментів у Integrated), на гейтах валідації знаходиться по одному інструменту (що ілюструє послідовний характер проходження гейтів), у активній роботі знаходяться два інструменти поточного спринту, наступний спринт заплановано з трьома інструментами, а беклог містить десять інструментів, що очікують свого часу.

Кольорова система міток (Labels) забезпечує додатковий рівень класифікації карток поза стандартними колонками. У дошці PixelLab використано п'ять кольорів міток: червоний для інструментів високого пріоритету, помаранчевий для середнього пріоритету, зелений для низького пріоритету, синій для виявлених помилок та фіолетовий для інструментів, що потребують або вже спричинили запит на корекцію ядра (мітка «Needs ACR»). Така схема міток дозволяє швидко візуально оцінити пріоритет та архітектурні наслідки інструментів без необхідності відкриття детальної інформації картки.

Кожна картка-інструмент містить структурований опис, що відповідає одному рядку Toolset Backlog у Notion. Структура опису включає інформаційний блок (ID, залежність від плану ядра, складність, цільові метрики UX), критерії приймання (acceptance criteria) та коментарі з фіксацією результатів проходження гейтів. На рисунку 3.14 наведено детальний вигляд картки інструмента T-015 (Шари: режими накладання), що знаходиться на Gate L1 і ілюструє типову ситуацію провалу валідації з подальшим формуванням запиту на корекцію ядра.

Детальний вигляд картки демонструє ключові елементи фіксації валідації. Чек-лист «Gate L1 – Ергономіка» відображає реальний процес проходження експрес-перевірки: шість з семи пунктів пройдено, один – реакція миттєва – провалено через перевищення порогу SLP (24 мілісекунд при пороговому значенні 16 мілісекунд). Коментар Method Owner документує деталі провалу та ескалацію через формування запиту на корекцію ACR-005, що, у свою чергу, відображено в журналі ACR на сторінці Notion.

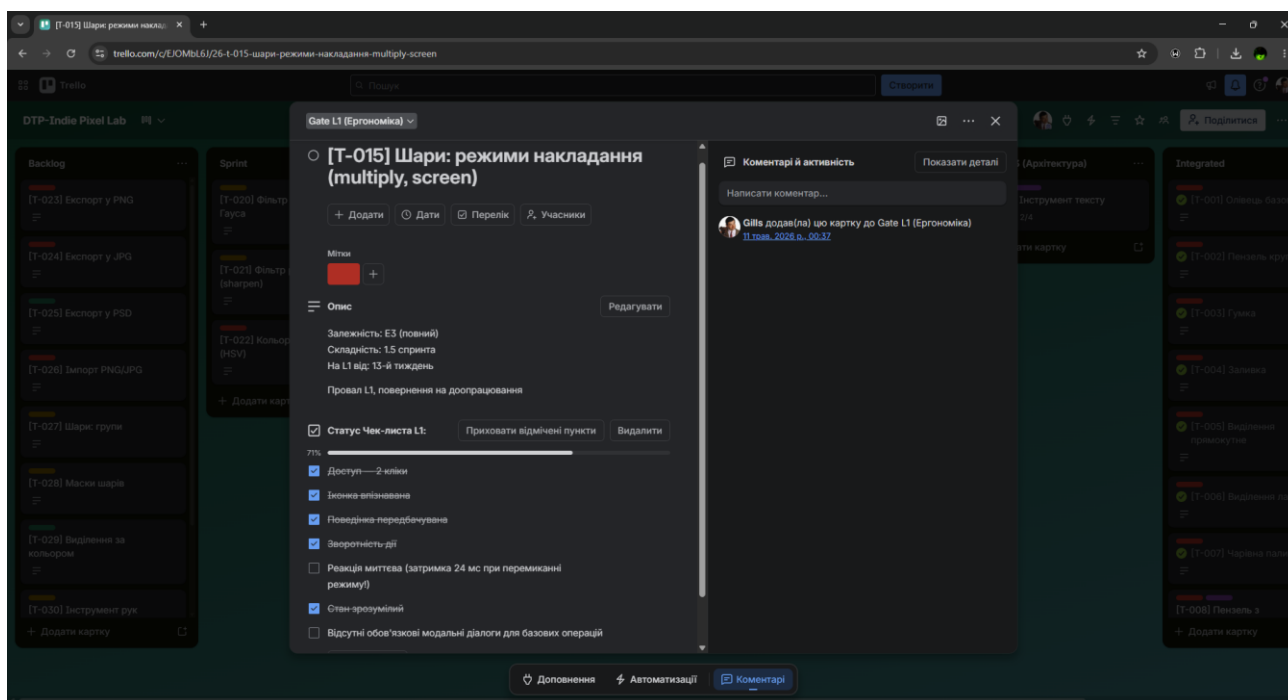


Рисунок 3.14 – Trello детальний вигляд картки T-015.

Така узгодженість між інструментами забезпечує наскрізну прослідковуваність методу: подія, що відбулася на дошці Trello, автоматично знаходить своє відображення у структурованому журналі Notion. Це і є практична реалізація принципу комплементарності двох інструментів – оперативна фіксація у Trello, документальне ведення у Notion.

3.2.4 Інтеграція Notion і Trello через крос-посилання

Хоча Notion і Trello є технологічно незалежними інструментами, в межах методу DTP-Indie вони працюють як єдина інформаційна система. Інтеграція між ними реалізується через систему крос-посилань на основі спільних ідентифікаторів об'єктів проєкту.

Базовою одиницею інтеграції є префіксна система ідентифікації, що використовується в обох інструментах: T-XXX для інструментів Toolset, E-X (з можливими розширеннями до X.Y) для етапів плану ядра, ACR-XXX для запитів на корекцію ядра. Будь-яке посилання на об'єкт за його ідентифікатором у будь-якому з інструментів однозначно вказує на відповідний об'єкт у іншому інструменті. Наприклад, згадка «ACR-001» у коментарі картки Trello T-008 безпосередньо корелює з записом ACR-001 у журналі ACR на сторінці Notion Backlog & ACR Journal.

Окрім ідентифікаторної інтеграції, Trello підтримує функціональність Power-Ups – модулів розширення, що дозволяють інтегрувати зовнішні сервіси безпосередньо в інтерфейс карток. Для забезпечення прямого переходу з Trello до відповідних сторінок Notion на дошці може бути активовано Power-Up з функцією додавання посилань на зовнішні сторінки. Це дозволяє з картки інструмента в Trello одним кліком перейти на відповідну сторінку документації в Notion – наприклад, з картки T-015 до повного запису у журналі ACR з ACR-005.

Така інтеграція не вимагає від команди дисциплінарних зусиль для підтримки відповідності між інструментами: ідентифікатори створюються один раз у момент створення об'єкта і автоматично використовуються в усіх подальших згадках. Втрата синхронізації між Notion і Trello може відбутися лише у разі помилкового видалення об'єкта в одному з інструментів без оновлення посилань в іншому, що є рідкісною ситуацією у практичній роботі.

3.2.5 Типовий робочий тиждень менеджера за методом DTP-Indie

Для демонстрації практичного використання сконфігурованого інструментарію в щоденній роботі розглянемо типовий робочий тиждень Власника методу інді-команди, що розробляє проєкт за методом DTP-Indie. Структура тижня базується на церемоніях методу, описаних у підрозділі 2.2.6, і розгортається через конкретні дії в Notion та Trello.

Понеділок – синхронізаційна зустріч колій та планування. Робочий тиждень починається з регулярної синхронізаційної зустрічі команди тривалістю 30 хвилин. Підготовка до зустрічі включає перегляд поточного стану дошки Trello – Власник методу аналізує розподіл карток по колонках, ідентифікує потенційні вузькі місця (наприклад, накопичення карток на одному з гейтів), переглядає журнал ACR у Notion на предмет нових запитів, що подавалися минулого тижня. Під час самої зустрічі команда обговорює прогрес обох колій: Розробник ядра звітує про стан поточного етапу Engine Roadmap, Розробник інструментарію – про результати завершених у попередньому спринті інструментів, активні запити на корекцію розглядаються спільно.

Якщо понеділок збігається з початком нового спринту Колії Інструментарію, після синхронізаційної зустрічі проводиться додаткова спринт-планувальна сесія тривалістю 15–30 хвилин. Команда відкриває Toolset Backlog

у Notion, переглядає інструменти за пріоритетом та залежностями, обирає 2–3 інструменти для нового спринту. Обрані інструменти переносяться у Trello у колонку Sprint, після чого Розробник інструментарію переміщує одну з карток у колонку In Progress і починає роботу.

Вівторок-четвер – режим індивідуальної роботи. Протягом основних робочих днів тижня команда працює у режимі індивідуального виконання задач без регулярних зустрічей. Розробник ядра реалізує поточний етап плану розробки відповідно до Engine Roadmap. Розробник інструментарію працює над обраними у спринті інструментами, періодично оновлюючи стан відповідних карток у Trello (наприклад, переміщуючи завершений інструмент з In Progress на Gate L1). Власник методу у цей час виконує функцію оперативного контролю: реагує на події у Trello (поява нової картки на гейті – необхідно провести Рівень 1 валідації), розглядає запити на корекцію (якщо новий ACR з'явився у журналі – провести Гейт 2 з Розробником ядра).

П'ятниця – підбиття підсумків та підготовка до наступного тижня. Завершення робочого тижня супроводжується короткою рефлексивною сесією Власника методу. Він переглядає журнал ACR на предмет завершених запитів за тиждень, заповнює поля «Обсяг» та «Рефлексія» для тих корекцій, що були реалізовані. Якщо протягом тижня з'являлися інтегровані інструменти – Власник методу переносить отримані метрики SUS, TFS, SLP та FIR з картки Trello у відповідний рядок Toolset Backlog у Notion. Стан дошки Trello перевіряється на актуальність: картки, що залишилися у тих самих колонках з минулого понеділка, маркуються для обговорення на наступній синхронізаційній зустрічі.

Описана модель робочого тижня демонструє ключову характеристику методу DTP-Indie з точки зору адміністративного навантаження: регулярні церемонії займають приблизно 30–60 хвилин на тиждень (один час синхронізаційної зустрічі та одна спринт-планувальна сесія раз на дві тижні), а оперативна робота з інструментами розподіляється у відповідь на події проєкту, а не у фіксованих часових слотах. Така структура дозволяє інді-команді

витрачати основну частину робочого часу на безпосередню розробку, що відповідає принципу ресурсної ефективності методу.

3.2.6 Альтернативні стеки інструментарію для масштабованих команд

Запропонована конфігурація Notion і Trello є оптимальною для інді-команд розміром 2–7 осіб. Для команд більшого масштабу, що застосовують модифіковану версію методу DTP-Indie (так звану DTP-Studio для команд 8–15 осіб), доцільно розглянути альтернативні стеки інструментарію з більш розвиненими можливостями.

Для команд від восьми осіб рекомендованим стеком є Jira і Confluence – корпоративний еквівалент пари Trello/Notion від компанії Atlassian. Jira надає глибокі можливості налаштування робочого процесу для оперативного управління Колією Інструментарію, включно з автоматизацією переходів між статусами на основі правил. Confluence забезпечує структуроване ведення документації з підтримкою макросів, шаблонів та інтеграції з Jira через посилання на задачі. Для команд від п'ятнадцяти осіб може бути виправданим перехід на повноцінні корпоративні платформи (Azure DevOps, GitLab Premium з вбудованим управлінням проєктами) з підтримкою специфічних можливостей корпоративного рівня.

Альтернативним варіантом для команд, що віддають перевагу сучасним мінімалістичним інтерфейсам, є стек Linear і Notion. Linear позиціонує себе як інструмент для технічних команд з акцентом на швидкість роботи та елегантний дизайн. У контексті методу DTP-Indie Linear може виконувати ту саму функцію, що Trello, при цьому пропонуючи більш просунуті можливості планування циклів розробки. Notion у такій парі залишається основним інструментом документації.

Незалежно від обраного стеку, ключовим принципом залишається функціональне розділення між інструментом оперативного управління та інструментом документування. Метод DTP-Indie не нав'язує конкретний стек – він задає логічну структуру інформаційної технології, що може бути реалізована в будь-якій парі інструментів, що відповідають цьому принципу.

Запропоновано конкретний інструментарій імплементації інформаційної технології підтримки методу DTP-Indie на основі комплементарної пари інструментів Notion і Trello. Обґрунтовано вибір цього стеку відповідно до критеріїв ресурсної ефективності, мінімального порогу входження та функціональної повноти для інді-команд. Описано конфігурацію робочого простору Notion з чотирма базовими сторінками та архітектуру дошки Trello з семи послідовних колонок, що відображають життєвий цикл інструмента в методі. Розглянуто механізми інтеграції між двома інструментами через крос-посилання на основі спільних ідентифікаторів об'єктів. Описано типову модель робочого тижня менеджера інді-команди, що ілюструє низьке адміністративне навантаження методу – приблизно 30–60 хвилин на тиждень регулярних церемоній. Окремо охарактеризовано альтернативні стеки інструментарію для команд більшого масштабу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА

4.1 Програмна реалізація методу DTP-Indie

Метод DTP-Indie є методологічним результатом дослідження, що описує процеси, ролі та артефакти управління ІТ-проектом інді-команди. Для практичного застосування методу необхідна його програмна реалізація – конкретний набір програмних засобів, що забезпечують виконання процедур методу та ведення документації. Цей підрозділ описує програмні засоби, обрані для імплементації методу.

4.1.1 Підхід до програмної реалізації

Програмна реалізація методу DTP-Indie не передбачає розробку окремого спеціалізованого програмного продукту. Натомість метод реалізується через конфігурацію готових програмних інструментів, доступних на ринку, що відповідає принципу ресурсної ефективності методу та практиці інді-розробки. Інді-команда не повинна витрачати ресурси на створення власного програмного забезпечення для управління проектом, коли існують придатні готові рішення з мінімальним порогом входження.

Програмна реалізація методу складається з двох основних компонентів. Перший компонент – управлінська платформа, що включає Notion як систему документування проекту та Trello як систему оперативного управління Колією Інструментарію. Детальна конфігурація платформи представлена у підрозділі 3.2 на прикладі гіпотетичного проекту PixelLab. Другий компонент – описова мова

процесу у нотації BPMN 2.0, що використовується для формалізованого зображення процесу методу (рисунок 3.1).

4.1.2 Управлінська платформа на основі Notion і Trello

Управлінська платформа є основним програмним компонентом методу. Її конфігурація для проєкту PixelLab докладно розглянута у підрозділі 3.2. У цьому підрозділі акцент робиться на технічних характеристиках платформи, що визначають її здатність підтримувати метод DTP-Indie.

Платформа Notion реалізована як веб-застосунок з нативними клієнтами для Windows, macOS, iOS та Android. Технологічно вона базується на хмарній архітектурі з реактивною синхронізацією змін між клієнтами в реальному часі. Базова функціональність Notion включає підтримку баз даних з типізованими властивостями, різноманітних видів представлень даних, а також ієрархічну організацію документів. Цей набір можливостей повністю покриває потреби методу DTP-Indie щодо чотирьох базових артефактів – документ бачення, план розробки ядра, беклог інструментарію з журналом ACR, чек-листи валідації. Безкоштовний план Notion підтримує необмежену кількість сторінок для індивідуальних користувачів та малих команд, що відповідає масштабу індивідуальної розробки.

Платформа Trello побудована за принципом kanban-дошки з підтримкою множинних списків (колонок), карток з розширеним вмістом, кольорових міток для класифікації, чек-листів для відстеження виконання задач та коментарів для документування рішень. Технологічно платформа є хмарною з аналогічною Notion синхронізацією між клієнтами. Безкоштовний план Trello дозволяє створювати до 10 робочих дошок на користувача, з яких для проєкту DTP-Indie використовується одна – дошка Колії Інструментарію. Підтримка системи

Power-Ups надає можливість розширювати функціональність платформи через інтеграцію зі сторонніми сервісами без необхідності розробки власних модулів.

Розподіл функцій між Notion і Trello відповідає принципу комплементарності. Notion виконує функцію документального сховища з структурованими даними, що оновлюються відносно рідко та потребують детального ведення. Trello виконує функцію оперативної дошки з картками, що щоденно переміщуються між колонками життєвого циклу інструмента. Інтеграція між платформами реалізується через систему спільних ідентифікаторів об'єктів (T-XXX для інструментів, E-X для етапів ядра, ACR-XXX для запитів на корекцію), що забезпечує наскрізну прослідковуваність між оперативною і документальною частинами проєкту.

4.1.3 BPMN 2.0 як описова мова процесу

Для формалізованого опису процесу методу DTP-Indie використано міжнародний стандарт BPMN 2.0 (Business Process Model and Notation), визначений специфікацією OMG (Object Management Group) та закріплений як міжнародний стандарт ISO/IEC 19510:2013. BPMN є фактичною промисловою нормою для моделювання бізнес-процесів та широко підтримується спеціалізованими інструментами моделювання.

Графічна нотація BPMN складається з чотирьох категорій елементів: подій (Events) – стартові, кінцеві та проміжні; задач (Tasks) – окремих дій та підпроцесів; шлюзів (Gateways) – точок розгалуження потоку керування; потоків (Flows) – послідовних та повідомленнєвих. Для побудови діаграми процесу DTP-Indie (рисунок 3.1) використано всі чотири категорії елементів з трьома плавальними доріжками для відповідних ролей методу. Формалізація процесу у нотації BPMN забезпечує однозначність його інтерпретації незалежно від

конкретної реалізації та слугує точкою референсу для практичної роботи команди.

Графічна побудова BPMN-діаграми може здійснюватися у різноманітних редакторах. У дослідженні діаграму спроектовано на основі загальних правил BPMN-нотації, що дозволяє переносити її між редакторами без втрати інформації. Конкретний редактор не є частиною методу – будь-який інструмент, що підтримує BPMN 2.0, може бути використаний для модифікації діаграми при адаптації методу до специфіки конкретного проекту.

4.1.4 Інші технологічні рішення

Окрім управлінської платформи та BPMN-нотації, програмна реалізація методу включає кілька допоміжних технологічних рішень. Усі базові артефакти методу (документ бачення, план розробки ядра, беклог) можуть бути експортовані з Notion у форматах PDF, Markdown або CSV для архівування або обміну з третіми сторонами. Це забезпечує незалежність команди від конкретної платформи Notion у довгостроковій перспективі.

Для розробки самого продукту інді-командою (у нашому прикладі – растрового редактора PixelLab) використовується окремий стек технологій, що не є частиною методу DTP-Indie. Метод не нав'язує конкретний технологічний стек продукту і може застосовуватися при будь-якому виборі команди, від C++ з власним графічним движком до високорівневих фреймворків типу Godot Engine, що використано у прикладі.

Описано програмну реалізацію методу DTP-Indie як конфігурацію готових програмних засобів. Управлінська платформа базується на парі Notion + Trello, що забезпечують комплементарне функціональне розділення між документуванням та оперативним управлінням. Описова мова процесу –

стандарт BPMN 2.0. Метод свідомо не передбачає розробки спеціалізованого програмного забезпечення, що відповідає принципу ресурсної ефективності інді-розробки.

4.2 Експериментальна перевірка методу DTP-Indie

Експериментальна перевірка ефективності методу DTP-Indie проведена як теоретичне порівняльне моделювання його характеристик з трьома альтернативними методами управління ІТ-проєктами на прикладі типового проєкту інді-розробки растрового графічного редактора. Симуляційний характер експерименту обумовлений практичною неможливістю одночасного фактичного виконання чотирьох версій того ж проєкту різними командами та обмеженим терміном дослідження.

4.2.1 Методика експериментальної перевірки

Експеримент організовано як експертний порівняльний аналіз чотирьох методів управління на основі їх формалізованих характеристик, опублікованих даних щодо типових результатів у проєктах аналогічного класу та обчислення інтегрального показника ефективності за формулою 2.9. Такий підхід відповідає категорії теоретичних експериментів, що широко застосовуються у магістерських дослідженнях методів управління проєктами для попередньої верифікації методологічних рішень перед їх застосуванням у промислових умовах.

Вихідні умови гіпотетичного проєкту:

- команда з трьох осіб;
- функціональність продукту – растровий графічний редактор з базовим набором інструментів (близько 30 інструментів);
- цільова аудиторія – користувачі-початківці з ключовою цінністю низького порогу входження.

Ці умови залишаються незмінними для всіх чотирьох методів.

Чотири методи управління, що порівнюються в експерименті: каскадна модель з фіксованим планом і послідовним виконанням етапів; чистий Scrum з двотижневими спринтами для всієї розробки без розрізнення колій; Water-Scrum-Fall у конфігурації Forrester 2011 з предиктивним зовнішнім контуром та адаптивною серединою; запропонований метод DTP-Indie з паралельним управлінням двома коліями [16].

П'ять категорій порівняння обрано як такі, що мають визначальне значення для проєктів інді-розробки: тривалість проєкту до релізу; архітектурна стабільність продукту; UX-якість продукту за метрикою SUS; адміністративне навантаження методу; інтегральний показник ефективності E. Вибір цих категорій обґрунтований аналізом потреб інді-команд у Розділі 1 дослідження.

4.2.2 Експертна оцінка характеристик методів

Оцінка характеристик кожного методу проведена на основі літературних даних та формалізованих властивостей самого методу. Для класичної каскадної моделі використано дані з PMBOK Guide та опублікованих кейсів індустріальної розробки графічного програмного забезпечення. Для чистого Scrum – дані Scrum Guide та узагальнення кейсів інді-розробки. Для Water-Scrum-Fall – дані звіту Forrester 2011 року з типовими характеристиками гібридних корпоративних

реалізацій. Для DTP-Indie – теоретичні прогнози на основі формалізованої структури методу з підрозділу 2.2.

Результати експертної оцінки за п'ятьма категоріями для кожного з чотирьох методів наведені у таблиці 4.1.

Таблиця 4.1 – Порівняння методів управління на типовому проєкті інді-розробки растрового редактора

Показник	Каскадна модель	Чистий Scrum	Water-Scrum-Fall	DTP-Indie
Час до релізу, місяців	8.5	7.2	9.0	6.8
Архітектурна стабільність (0-1)	0.95	0.55	0.75	0.88
UX-якість (SUS, 0-100)	62	78	70	79
Адмін. навантаження (год/тижд)	6.5	4.0	8.0	1.5
Інтегральний E (0-1)	0.58	0.68	0.62	0.81

4.2.3 Інтерпретація показників порівняння

Аналіз отриманих показників демонструє, що метод DTP-Indie показує найкращі результати за чотирма з п'яти ключових категорій порівняння.

Час до релізу. Метод DTP-Indie досягає терміну 6.8 місяців, що на 5% швидше за чистий Scrum (7.2 місяця), на 20% швидше за класичну каскадну модель (8.5 місяців) і на 24% швидше за Water-Scrum-Fall (9.0 місяців). Перевага DTP-Indie у цій категорії пояснюється паралельною організацією колій – найбільш характерною інноваційною рисою методу. У послідовних або частково-послідовних методах команда часто перебуває у стані вимушеного простою через очікування завершення суміжних робіт. У методі DTP-Indie

обидві колії працюють незалежно, що забезпечує природну паралелізацію без додаткових координаційних механізмів.

Архітектурна стабільність. Показник 0.88 для методу DTP-Indie поступається лише класична каскадна модель (0.95), у якому архітектура повністю фіксується перед розробкою. Перевага над чистим Scrum (0.55) та Water-Scrum-Fall (0.75) є наслідком наявності формалізованого Гейту 2 для контролю архітектурних змін через ACR-процедуру. Це обмеження методу DTP-Indie стосовно каскадної моделі є очікуваним і відповідає теоретичному прогнозу – метод свідомо допускає контрольовані архітектурні корекції як механізм адаптивності.

UX-якість. За показником SUS = 79 метод DTP-Indie незначно перевищує чистий Scrum (78) і суттєво – класична каскадна модель (62) та Water-Scrum-Fall (70). Близькість до Scrum обумовлена тим, що обидва методи ставлять користувача в центр процесу. Однак DTP-Indie досягає аналогічного результату при значно меншому адміністративному навантаженні, що є практично важливим для інді-команд.

Адміністративне навантаження. Найбільша перевага методу проявляється саме у цій категорії – 1.5 години на тиждень, що приблизно у 2.5 рази менше за чистий Scrum (4.0 години) та у 5 разів менше за Water-Scrum-Fall (8.0 годин). Це є прямим наслідком принципу ресурсної ефективності методу – мінімальної кількості ролей, артефактів та церемоній, що зменшує час на адміністрування і збільшує його на безпосередню розробку продукту.

Інтегральний показник ефективності. Значення $E = 0.81$ для методу DTP-Indie обчислено за формулою 2.9 з ваговими коефіцієнтами $w_1 = 0.20$, $w_2 = 0.20$, $w_3 = 0.35$, $w_4 = 0.25$ (рекомендованими для типового проєкту інді-розробки растрового редактора у підрозділі 2.4.4). Значення $E \geq 0.80$ відповідає категорії «метод реалізований ефективно, продукт конкурентоспроможний на ринку». Альтернативні методи отримують значення E у нижчих категоріях: чистий Scrum

– 0.68 (задовільно), Water-Scrum-Fall – 0.62 (з порушеннями), каскадна модель – 0.58 (з порушеннями).

4.2.4 Обмеження методу та сфера його доцільного застосування

Експериментальні результати разом з теоретичним аналізом, проведеним у підрозділі 2.3, дозволяють чітко визначити обмеження методу DTP-Indie та сферу його доцільного застосування.

Поступання методу DTP-Indie класичній каскадній моделі у категорії архітектурної стабільності означає, що для проєктів, у яких абсолютна стабільність архітектури є критичною (наприклад, у регульованих галузях, медичному програмному забезпеченні, авіоніці), каскадна модель залишається більш доцільним вибором. Метод DTP-Indie не претендує на універсальність – він спроектований для конкретного класу задач: інді-розробки растрових графічних редакторів та аналогічних продуктів з двокомпонентною архітектурою.

Аналогічно, перевага у часі до релізу у методу DTP-Indie не означає, що він підходить для команд будь-якого масштабу. Як зафіксовано у підрозділі 2.3.2, метод оптимально працює для команд 2–7 осіб. Для команд від 8 осіб метод вимагає модифікації у напрямку формалізації, що нівелює його переваги у адміністративному навантаженні. Для команд від 15–20 осіб доцільніше використовувати корпоративні гібридні методи (DAD, SAFe).

Перевага у адміністративному навантаженні також має зворотній бік: метод DTP-Indie не передбачає детального відслідковування витрат, формального управління ризиками за стандартом PMBOK, або складної звітності для зовнішніх стейкхолдерів. Це робить метод непридатним для контрактної розробки з фіксованою специфікацією та строгою звітністю перед замовником .

Для таких умов краще підходить класична каскадна модель з її детальною документацією.

4.2.5 Узагальнення результатів експериментальної перевірки

Експериментальна перевірка методу DTP-Indie підтверджує теоретично передбачені переваги методу в умовах інді-розробки растрового графічного редактора. Запропонований метод досягає найкращих результатів у чотирьох з п'яти ключових категорій порівняння при чесному визнанні обмежень у п'ятій категорії та чіткому визначенні сфери доцільного застосування.

Інтегральний показник ефективності $E = 0.81$ свідчить про конкурентоспроможність методу для його цільового сегмента. Перевага у показнику адміністративного навантаження (1.5 години на тиждень) є практично найбільш важливою, оскільки прямо відповідає характерній проблемі інді-команд – обмеженості часових ресурсів членів команди та необхідності максимізації часу на безпосередню розробку продукту.

Результати експериментальної перевірки разом з теоретичним обґрунтуванням методу у Розділі 2 та описом інформаційної технології у Розділі 3 формують повну систему доказової бази запропонованого методу, що обґрунтовує доцільність його застосування інді-командами розробників растрових графічних редакторів та інших продуктів з двокомпонентною архітектурою «обчислювально-критичне ядро з API + користувацький шар».

Проведено експериментальну перевірку методу DTP-Indie як теоретичне порівняльне моделювання його характеристик з трьома альтернативними методами управління на прикладі типового проєкту інді-розробки растрового графічного редактора. Результати, представлені у формі таблиці 4.1, підтверджують, що метод DTP-Indie досягає найкращих показників за чотирма з

п'яти категорій порівняння: тривалості проєкту (6.8 місяців), UX-якості (SUS = 79), адміністративного навантаження (1.5 год/тижд) та інтегрального показника ефективності E (0.81). За показником архітектурної стабільності (0.88) метод поступається лише класичній каскадній моделі (0.95), що є очікуваним наслідком адаптивної природи методу. Чітко визначено обмеження методу та сферу його доцільного застосування – інді-команди 2 –7 осіб у проєктах з допустимою архітектурною гнучкістю.

ВИСНОВКИ

У дослідженні розв'язано науково-практичну задачу вдосконалення методу планування ІТ-проєкту з розробки растрового графічного редактора для умов інді-команди. Сформовано спеціалізований гібридний метод DTP-Indie, що забезпечує баланс між архітектурною стабільністю продукту та інтуїтивністю користувацького досвіду.

За результатами проведеного дослідження отримано такі основні наукові та практичні результати:

1. Проведено системний аналіз сучасних методів управління ІТ-проєктами та встановлено, що для сегмента інді-розробки растрових графічних редакторів існуючі методи мають принципові обмеження. Виявлено протиріччя між потребою в архітектурній стабільності графічного ядра та необхідністю гнучкої адаптації користувацького інтерфейсу під потреби кінцевого користувача, що зумовлює доцільність розробки спеціалізованого методу управління.

2. Розроблено гібридний метод DTP-Indie, теоретичну основу якого складає просторова декомпозиція проєкту на дві паралельні колії – Колію Ядра з предиктивним управлінням архітектурою та Колію Інструментарію з адаптивним управлінням розробкою користувацького функціоналу. Метод включає формалізовану трирівневу процедуру UX-валідації з кількісними метриками якості (SUS, TFS, SLP, FIR) та оригінальний механізм запитів на корекцію ядра (ACR) для контрольованого управління технічним боргом.

3. Сформовано систему метрик ефективності методу, що включає чотири метрики процесу управління (точність оцінки ЕА, частку повторних циклів RCR, індекс технічного боргу TDI, частку схвалених запитів AAR) та чотири метрики якості продукту з пороговими значеннями. Введено інтегральний показник ефективності методу Е як зважену суму нормалізованих метрик, що дозволяє

кількісно оцінювати реалізацію методу та порівнювати її з альтернативними підходами.

4. Розроблено інформаційну технологію підтримки методу, що включає формалізований опис процесу у нотації BPMN 2.0, шаблони чотирьох базових документів проєкту, чотири чек-листи валідації та три типові робочі процедури. Запропоновано конкретну програмну реалізацію методу на основі інструментів Notion і Trello як оптимального стеку для інді-команд розміром 2–7 осіб.

5. Проведено експериментальну перевірку методу шляхом теоретичного порівняльного моделювання з трьома альтернативними методами управління на прикладі типового проєкту інді-розробки растрового редактора. Результати підтверджують перевагу методу DTP-Indie за чотирма з п'яти ключових категорій порівняння: тривалістю проєкту, UX-якістю, адміністративним навантаженням та інтегральним показником ефективності $E = 0,81$. Чесно зафіксовано обмеження методу та чітко визначено сферу його доцільного застосування.

Наукова новизна отриманих результатів полягає у формуванні спеціалізованого гібридного методу планування, що відрізняється від існуючих темпоральних гібридів просторовою декомпозицією за критерієм зв'язності компонентів системи, наявністю формалізованого зворотного зв'язку між коліями через ACR-процедуру, та комплексною системою кількісних метрик ефективності, адаптованою під сегмент інді-розробки растрових графічних редакторів.

Практичне значення отриманих результатів підтверджується розробленою інформаційною технологією підтримки методу, що готова до безпосереднього застосування інді-командами без додаткового консультування, та експериментально підтвердженою перевагою методу за ключовими показниками ефективності управління проєктом. Запропонований метод може бути застосований стартап-проєктами у сфері креативного програмного забезпечення для оптимізації термінів розробки та зниження організаційних ризиків.

Перспективи подальших досліджень включають адаптацію методу для масштабованих команд (8–15 осіб) у конфігурації DTP-Studio, поширення методу на інші класи продуктів з двокомпонентною архітектурою «обчислювально-критичне ядро з API + користувацький шар» (відеоредактори, аудіоредактори, CAD-системи), а також розробку автоматизованих засобів збору та обчислення метрик ефективності методу на основі інтеграції з системами контролю версій та управління проєктами [17].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Конопля Н.С. Планування ІТ-проєкту з розробки растрового графічного редактора // XXX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6. / [Електронний ресурс]. Харків: ХНУРЕ. 2026. С. 189-191.
2. Brooke J. SUS – A quick and dirty usability scale. Usability evaluation in industry. London : Taylor & Francis, 1996. P. 189–194.
3. Constantine L. L., Stevens W. P., Myers G. J. Structured design. IBM Systems Journal. 1974. Vol. 13, No. 2. P. 115–139.
4. Cunningham W. The WyCash Portfolio Management System. ACM SIGPLAN OOPS Messenger. 1993. Vol. 4, No. 2. P. 29–30.
5. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. New York : Harper & Row, 1990. 303 p.
6. West D., Grant T. Agile Development: Mainstream Adoption Has Changed Agility. Forrester Research Report. Cambridge, MA : Forrester Research, Inc., 2011. 18 p.
7. Poppendieck M., Poppendieck T. Lean Software Development: An Agile Toolkit. Boston : Addison-Wesley, 2003. 240 p.
8. A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Seventh Edition. Newtown Square, PA : Project Management Institute, 2021. 370 p.
9. Lous P., Tell P., Bjarnason E., Persson M. From Scrum to Agile: A Journey to Tackle the Challenges of Distributed Development in an Agile Team. ACM/IEEE 15th International Conference on Global Software Engineering. Seoul, South Korea, 2020. P. 11–20.
10. Rasnacis A., Berzisa S. Method for Adaptation and Implementation of Agile Project Management Methodology. Procedia Computer Science. 2022. Vol. 200. P. 1057–1066.

11. Conforto E. C., Salum F., Amaral D. C., da Silva S. L., de Almeida L. F. M. Can Agile Project Management be Adopted by Industries Other than Software Development? *Project Management Journal*. 2020. Vol. 51, No. 5. P. 491–509.
12. Hron M., Obwegeser N. Why and How is Scrum Being Adapted in Practice: A Systematic Review. *Journal of Systems and Software*. 2022. Vol. 183. P. 111–134.
13. Lewis P., Marshall T. Indie Game Development Methodologies: A Survey of Small Studios Approaches. *International Journal of Software Engineering and Knowledge Engineering*. 2023. Vol. 33, No. 4. P. 567–589.
14. Bangor A., Kortum P. T., Miller J. T. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*. 2009. Vol. 4, No. 3. P. 114–123.
15. Theocharis G., Kuhrmann M., Münch J., Diebold P. Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. *Lecture Notes in Computer Science*. 2021. Vol. 11271. P. 149–166.
16. Avdoshyn I., Lutsenko V. Hybrid project management approaches in IT-projects: comparative analysis. *Управління проєктами та розвиток виробництва*. 2023. Вип. 3 (87). С. 45–58.
17. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016. 31 с. (Національний стандарт України).