МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Державне некомерційне підприємство «Державний університет»
Київський авіаційний інститут

Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувачка кафедри
Олена ГРІНЕНКО

"____"_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: " Адаптивний алгоритм та застосунок для аналізу ефективності часу розробки програмного забезпечення "

Виконавець: Драгусевич Назар Ігорович

Керівник: к.т.н доцент Шибицька Наталія Миколаївна

Нормоконтролер:

Київ 2025

ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО «ДЕРЖАВНИЙ УНІВЕРСИТЕТ» КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ

Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувачка кафедри
Олена ГРІНЕНКО
"___" _____ 2025 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Драгусевича Назара Ігоровича

1. Тема роботи: «Адаптивний алгоритм та застосунок для аналізу ефективності часу розробки програмного забезпечення» затверджена наказом ректора від 2025р. № 678/ст

2. Термін виконання роботи: з 06.09.2025 р. до 30.11.2025 р.

3. Вихідні данні до роботи: веб-застосунок для управління часом та аналізу продуктивності розробників програмного забезпечення.

4. Зміст пояснювальної записки:
1. Аналіз проблеми управління часом у розробці програмного забезпечення.
2. Проєктування адаптивної системи управління часом та аналізу продуктивності
3. Реалізація застосунку та експериментальне дослідження ефективності.
4. Висновки та список використаних джерел.

5. Перелік обов'язкових слайдів презентації:

- Актуальність проблеми управління часом у розробці ПЗ
- Огляд існуючих методологій та інструментів
- Математична модель адаптивного алгоритму
- Методологія експериментального дослідження
- Результати експериментів
- Висновки та перспективи розвитку

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та вивчення літератури Написання 1 розділу, представлення керівнику	06.09-12.09	
2.	Попередній друк 1 розділу та допоміжних сторінок (черновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел. 1-ий нормо-контроль.	13.09-30.09	
3.	Написання 2 розділу, представлення керівнику	1.10-24.10	
4.	Написання 3 розділу, представлення керівнику	25.10-23.11	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	24.11-30.11	
7.	Проходження нормо-контролю, перепліт пояснювальної записки.	01.12-08.12	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	09.12-10.12	
9.	Отримання відгуку керівника, рецензії.	11.12-12.12	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 2 папки, 2 конверта)	13.12-14.12	

7. Дата видачі завдання 06.09.2025

Керівник: _____ к.т.н. доцент Наталія ШИБИЦЬКА

Завдання прийняв до виконання: _____ Назар ДРАГУСЕВИЧ

Дата 06.09.2025

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Адаптивний алгоритм та застосунок для аналізу ефективності часу розробки програмного забезпечення»: 94 с., 29 рис., 25 табл., 60 інформаційних джерел.

УПРАВЛІННЯ ЧАСОМ, ПРОДУКТИВНІСТЬ РОЗРОБНИКІВ, АДАПТИВНИЙ АЛГОРИТМ, ТРЕКІНГ ЧАСУ, АНАЛІТИКА, BLAZOR WEBASSEMBLY, ASP.NET CORE, ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ.

Об’єкт розробки - адаптивна система управління часом та аналізу продуктивності для розробників програмного забезпечення з функціями автоматичного трекінгу, аналітики метрик та персоналізованих рекомендацій.

Мета роботи – підвищення ефективності роботи розробників програмного забезпечення шляхом розробки та експериментальної верифікації адаптивної системи управління часом з інтелектуальними алгоритмами аналізу індивідуальних паттернів продуктивності.

ABSTRACT

Explanatory note to the qualification work "Adaptive Algorithm and Application for Analyzing Software Development Time Efficiency": 94 p., 29 figures, 25 table, 60 information sources.

TIME MANAGEMENT, DEVELOPER PRODUCTIVITY, ADAPTIVE ALGORITHM, TIME TRACKING, ANALYTICS, BLAZOR WEBASSEMBLY, ASP.NET CORE, EXPERIMENTAL STUDY.

Property development - an adaptive time management and productivity analysis system for software developers with automatic tracking, metrics analytics, and personalized recommendations.

Purpose - to improve the efficiency of software developers through the development and experimental verification of an adaptive time management system with intelligent algorithms for analyzing individual productivity patterns.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

API – Application Programming Interface (інтерфейс програмування додатків)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (користувацький досвід)

БД – база даних

SQL – Structured Query Language (структурована мова запитів)

ASP.NET – платформа веб-розробки від Microsoft

EF – Entity Framework (фреймворк для роботи з базами даних)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

IDE – Integrated Development Environment (інтегроване середовище розробки)

CI/CD – Continuous Integration/Continuous Deployment (безперервна інтеграція/безперервне розгортання)

SDLC – Software Development Life Cycle (життєвий цикл розробки програмного забезпечення)

KPI – Key Performance Indicator (ключовий показник ефективності)

MVP – Minimum Viable Product (мінімально життєздатний продукт)

REST – Representational State Transfer (архітектурний стиль для API)

MVC – Model-View-Controller (архітектурний патерн)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

LINQ – Language Integrated Query (інтегрована в мову система запитів)

URL – Uniform Resource Locator (уніфікований локатор ресурсу)

SPA – Single Page Application (односторінковий додаток)

GUI – Graphical User Interface (графічний інтерфейс користувача)

ВСТУП

Актуальність теми. Розробка програмного забезпечення є інтелектуально затратною діяльністю, що вимагає ефективного управління часом. За даними Stack Overflow Developer Survey 2024, понад 68% розробників відзначають проблеми з продуктивністю через неефективний розподіл часу та відсутність систематичного підходу до планування. Дослідження Atlassian показує, що середній розробник витрачає близько 28% робочого часу на непродуктивну діяльність.

Неефективне управління часом має значний економічний вплив: перевитрата бюджетів на 23-27% та затримки у 45% проєктів (Project Management Institute). Для індивідуальних розробників втрата продуктивності може становити 15-40 годин на місяць.

Існуючі інструменти (RescueTime, Toggl, WakaTime) зосереджені на пасивному трекінгу, не враховують індивідуальні особливості та не надають персоналізованих рекомендацій. Актуальним є створення адаптивної системи, яка активно допомагає покращувати продуктивність через аналіз метрик та надання своєчасних рекомендацій.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана на кафедрі інженерії програмного забезпечення Національного авіаційного університету в рамках наукових досліджень кафедри з напрямку "Методи та засоби розробки програмного забезпечення".

Мета і завдання дослідження. Метою є підвищення ефективності використання часу розробників ПЗ шляхом розробки адаптивного алгоритму аналізу продуктивності та створення застосунку з персоналізованими рекомендаціями.

Завдання:

1. Проаналізувати існуючі методи та інструменти управління часом розробників ПЗ.

2. Дослідити метрики ефективності та визначити показники продуктивності.
3. Розробити математичну модель адаптивного алгоритму для розрахунку метрик продуктивності.
4. Спроєктувати архітектуру застосунку для управління завданнями та аналізу ефективності.
5. Реалізувати функціональні модулі системи.
6. Провести дослідження впливу факторів на продуктивність.
7. Виконати статистичний аналіз результатів та оцінити ефективність алгоритму.

Об'єкт дослідження – процес управління часом під час розробки ПЗ.

Предмет дослідження – методи, алгоритми та програмні засоби для аналізу ефективності використання часу розробників.

Методи дослідження. системний аналіз, математичне моделювання, об'єктно-орієнтоване проєктування, експериментальне дослідження, статистичний аналіз (t-критерій Стьюдента, ANOVA, кореляційний аналіз), методи програмної інженерії.

Наукова новизна:

1. Запропоновано адаптивний алгоритм, який враховує індивідуальні паттерни продуктивності та динамічно коригує метрики на основі історичних даних.
2. Розроблено математичну модель для кількісної оцінки впливу факторів на результативність роботи.
3. Експериментально підтверджено вплив чотирьох ключових факторів на продуктивність: перегляду статистики, сповіщень про перерви, дотримання інтервалів відпочинку та щоденного планування.

Практичне значення одержаних результатів. Створено функціонуючий веб-застосунок для управління часом з автоматичним трекінгом, персоналізованими рекомендаціями, візуалізацією статистики та системою

нагадувань. Результати можуть використовуватись IT-компаніями, менеджерами проєктів та інтегруватись у системи управління проєктами.

Особистий внесок здобувача. Автором особисто виконано аналіз предметної області, розроблено математичну модель та архітектуру, реалізовано всі модулі застосунку, проведено експериментальні дослідження та статистичний аналіз.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 94 сторінки, включаючи 29 рисунків, 25 таблиць. Список використаних джерел містить 59 найменувань. Додатки містять графічні матеріали, лістинги програмного коду, результати експериментів та анкети учасників дослідження.

У першому розділі проведено аналіз проблеми ефективного управління часом у розробці ПЗ, виконано огляд існуючих методів та інструментів, досліджено метрики продуктивності, розглянуто підходи до створення адаптивних систем та обґрунтовано необхідність розробки нового рішення.

У другому розділі визначено функціональні та нефункціональні вимоги до системи, розроблено математичну модель адаптивного алгоритму для розрахунку метрик продуктивності, спроєктовано архітектуру застосунку з використанням технологій ASP.NET, Entity Framework, Blazor, спроєктовано структуру бази даних, API та користувацький інтерфейс, обґрунтовано вибір технологій реалізації.

У третьому розділі описано реалізацію основних функціональних модулів застосунку, викладено методологію проведення експериментального дослідження, представлено результати експериментів щодо впливу різних факторів на продуктивність розробників, виконано аналіз даних, проведено оцінку ефективності адаптивного алгоритму та описано процес тестування.

У висновках підсумовано результати виконаної роботи, сформульовано основні наукові та практичні результати, визначено перспективи подальших досліджень.

РОЗДІЛ 1.
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ УПРАВЛІННЯ ЧАСОМ РОЗРОБНИКІВ ПЗ

1.1 Проблема ефективного використання часу в розробці програмного забезпечення

Розробка програмного забезпечення належить до знаннєво-інтенсивної діяльності, яка вимагає високого рівня когнітивних зусиль та глибокої концентрації. На відміну від інших видів професійної діяльності, продуктивність розробника не може бути оцінена простими метриками, оскільки якість архітектурних рішень та підтримуваність коду є важливішими факторами успіху.

Згідно з Stack Overflow Developer Survey 2024 (понад 90 тисяч респондентів), 68% розробників відзначають проблеми з продуктивністю через неефективне управління часом [1]. Основні проблеми: часті переривання (83%), неефективне планування (67%), багатозадачність (61%) та відсутність розуміння пріоритетів (54%).

Проблема переривань та перемикання контексту. Дослідження Глорії Марк показало, що після переривання розробнику потрібно 23 хвилини для відновлення концентрації [2]. При цьому середньостатистичний розробник зазнає переривань кожні 11 хвилин. Дослідження Atlassian виявило, що розробники витрачають до 31 години на місяць на непродуктивні зустрічі, а перемикання між завданнями знижує продуктивність на 20-40% [3].

Особливо руйнівним є ефект перемикання контексту для складних когнітивних завдань. У стані глибокої концентрації (flow state) продуктивність може бути в 5-10 разів вищою [4], однак досягнення цього стану вимагає 15-30 хвилин безперервної роботи, і будь-яке переривання негайно його порушує.

Проблема багатозадачності.

Нейрофізіологічні дослідження доводять, що людський мозок не здатний до справжньої багатозадачності – відбувається швидке перемикання уваги, що призводить до когнітивного навантаження [5]. Дослідження Американської психологічної асоціації показало, що багатозадачність знижує продуктивність на 40% та збільшує помилки у коді на 50% [6].

Методологія Kanban рекомендує працювати максимум над 1-2 завданнями одночасно. Компанії, які впровадили обмеження незавершеної роботи (WIP limits), відзначають підвищення пропускної здатності на 25-30% та скорочення cycle time на 35-45% [7].

Проблема відсутності систематичного планування. Лише 32% розробників регулярно використовують техніки планування [8]. Відсутність чіткого плану призводить до реактивного режиму роботи. Незаплановані завдання займають в середньому 45% робочого часу [9]. Чітке планування підвищує відчуття контролю, знижує стрес та збільшує мотивацію.

Розробники систематично недооцінюють час виконання завдань на 30-50% [10]. Ця "помилка планування" (planning fallacy) призводить до постійних затримок та перевантаження команди.

Економічні наслідки неефективного управління часом. За оцінками Project Management Institute, неефективне управління призводить до втрати 11.4% інвестицій в IT-проєкти [11]. Для IT-компанії з 50 розробників втрати становлять 500 тисяч – 1.5 мільйонів доларів на рік.

На індивідуальному рівні це призводить до переробок та професійного вигорання. Згідно з Gallup, лише 23% розробників відчувають себе продуктивними, тоді як 59% перебувають у стані емоційного вигорання [12]. Середній термін роботи на одному місці – 2.3 роки.

Вплив на якість програмного коду. Існує пряма кореляція між втомою та кількістю помилок [13]. Робота під тиском без достатніх перерв призводить до субоптимальних рішень та накопичення технічного боргу, який знижує швидкість розробки на 30-60% в довгостроковій перспективі [14]. Недостатня концентрація призводить до пропуску критичних уразливостей, виправлення яких на пізніх етапах коштує в 10-100 разів дорожче [15].

Фактори, що впливають на ефективність використання часу. Аналіз літератури та практичний досвід дозволяють виділити основні фактори, що впливають на ефективність використання часу розробників:

- **Організаційні**: культура зустрічей, процеси комунікації, система управління завданнями.
- **Технічні**: якість інструментів, швидкість збірки, стабільність інфраструктури.
- **Індивідуальні**: навички управління часом, рівень досвіду, здатність до концентрації.
- **Середовищні**: рівень шуму, наявність простору для глибокої роботи, ергономіка.

Найбільший вплив мають фактори, які можна контролювати індивідуально: планування, управління перериваннями, дотримання режиму відпочинку та використання інструментів для моніторингу роботи.

Проблема ефективного використання часу є комплексною та вимагає системного підходу. Створення інструментів для розуміння власних паттернів продуктивності є актуальним напрямком досліджень з значним практичним ефектом.

1.2 Огляд існуючих методів та інструментів управління часом розробників

Управління часом у розробці програмного забезпечення є предметом досліджень як в академічній літературі, так і в практичних методологіях. За

останні десятиліття було розроблено численні методи та інструменти для допомоги розробникам у ефективній організації робочого процесу.

1.2.1. Методології управління часом

Техніка Pomodoro. Розроблена Франческо Чірілло наприкінці 1980-х років [16]. Суть методу – розбиття робочого часу на 25-хвилинні інтервали ("помідори") з 5-хвилинними перервами та більш тривалими перервами (15-30 хвилин) після кожних 4 помідорів.

Алгоритм роботи з технікою Pomodoro показано на рисунку 1.1.

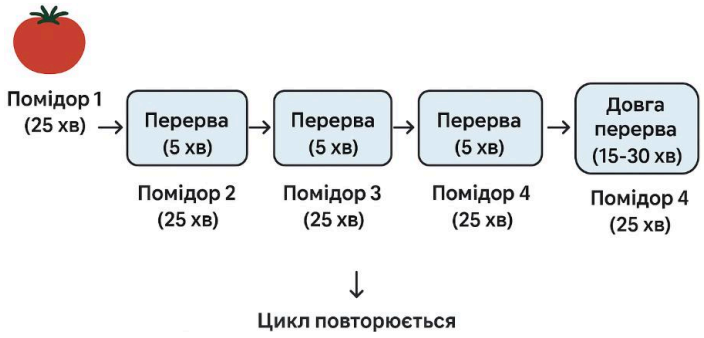

Рис. 1.1 – Схема роботи техніки Pomodoro

Переваги: простота, підтримка регулярних перерв, відчуття прогресу. Недоліки: фіксований 25-хвилинний інтервал недостатній для глибокої концентрації при складних завданнях [17], примусові переривання контрпродуктивні у потоковому стані.

Time Boxing. Виділення фіксованих блоків часу (від 30 хвилин до 4 годин) для конкретних завдань [18]. На відміну від Pomodoro, розмір блоку визначається індивідуально залежно від характеру завдання. Добре поєднується з календарним плануванням та активно використовується в Agile-методологіях.

Getting Things Done (GTD). Комплексна методологія управління завданнями Девіда Аллена [19]. Основні компоненти: збір всіх завдань, уточнення необхідних дій, організація за категоріями та контекстами,

регулярний огляд, виконання на основі контексту. Для розробників особливо корисна концепція контекстів (@computer, @code-review, @meeting).

Deep Work. Концепція Кала Ньюпорта підкреслює важливість тривалих періодів (90 хвилин – 4 години) безперервної концентрації на когнітивно складних завданнях [20]. Принципи: мінімізація відволікань, створення ритуалів для концентрації, розмежування глибокої та поверхневої роботи. Дослідження показують, що розробники, які практикують Deep Work, виконують складні завдання на 40-50% швидше [21]. Вимагає дисципліни та організаційної підтримки.

Порівняльний аналіз методологій. Таблиця 1.1 містить порівняльну характеристику розглянутих методологій управління часом з точки зору їх застосовності для розробників ПЗ.

Таблиця 1.1 Порівняльний аналіз методологій управління часом

Методологія	**Тривалість сесії**	**Частота перерв**	**Складність впровадження**	**Підходить для**	**Основний недолік**
Pomodoro	25 хв	Кожні 25 хв	Низька	Рутинні завдання, програмування за прикладом	Переривання потокового стану
Time Boxing	30 хв – 4 год	Між блоками	Середня	Планування спринтів, різнотипні завдання	Потребує хороших навичок оцінки часу
GTD	Не фіксовано	Не регламентовано	Висока	Управління множиною проєктів	Складність системи, час на підтримку

Як видно з таблиці, жодна з методологій не є універсальною для всіх типів завдань у розробці ПЗ. Оптимальний підхід часто включає комбінацію різних методів залежно від характеру роботи: Deep Work для складних завдань, що вимагають креативності, Pomodoro для рутинних задач, та Time Boxing для загального планування дня.

1.2.2. Інструменти автоматичного трекінгу часу

З розвитком технологій з'явилася можливість автоматизації процесу відстеження часу роботи розробників. Сучасні інструменти трекінгу часу можна класифікувати на кілька категорій залежно від рівня автоматизації та специфіки функціональності.

RescueTime – автоматичне відстеження активності на комп'ютері користувача [22]. Працює у фоновому режимі, записуючи час у додатках та веб-сайтах, автоматично категоризує діяльність, генерує звіти, дозволяє встановлювати цілі та блокувати відволікаючі сайти. Переваги: повністю автоматичний збір даних, детальна аналітика. Недоліки: неточність автоматичної категоризації для специфічних процесів розробників, обмежене розуміння контексту.

Toggl Track – мануальний трекінг з відносно простим інтерфейсом [23]. Розробник самостійно запускає/зупиняє таймер, вказуючи проєкт та опис. Організація за проєктами, тегами, інтеграція з Jira, GitHub. Популярний серед фрілансерів. Недолік: необхідність пам'ятати про запуск таймера.

WakaTime – є спеціалізованим інструментом для розробників, який інтегрується безпосередньо з IDE та текстовими редакторами [24]. Він автоматично відстежує час, проведений у коді, надаючи детальну статистику за мовами програмування, проєктами, файлами та навіть окремими функціями, якщо це необхідно.

Clockify - позиціонується як безкоштовна альтернатива Toggl з аналогічною функціональністю мануального трекінгу часу [25]. Відсутність

обмежень на кількість користувачів та проєктів у безкоштовній версії робить його популярним вибором для команд.

Таблиця 1.2 Порівняння інструментів трекінгу часу

Інструмент	**Тип трекінгу**	**Специфіка для розробників**	**Ціна (місяць)**	**Інтеграції**	**Основний недолік**
RescueTime	Автоматичний	Низька	$12	Обмежені	Неточна категоризація контексту
Toggl Track	Мануальний	Низька	$10	Широкі (Jira, GitHub)	Потрібно пам'ятати запускати
Інструмент	Тип трекінгу	Специфіка для розробників	Ціна (місяць)	Інтеграції	Основний недолік
WakaTime	Автоматичний	Висока	$9	IDE плагіни	Тільки час в IDE
Clockify	Мануальний	Низька	Безкоштовно	Широкі	Обмежена аналітика у безкоштовній версії

Аналіз показує, що існуючі інструменти трекінгу часу мають обмеження в контексті специфічних потреб розробників ПЗ. Автоматичні інструменти загального призначення не розуміють контексту розробки та можуть некоректно класифікувати діяльність. Спеціалізовані інструменти (WakaTime) відстежують лише частину робочого процесу. Мануальні інструменти (Toggl, Clockify) вимагають дисципліни та додаткових зусиль.

1.2.3. Системи аналізу продуктивності розробників

GitHub Insights та GitLab Analytics – аналітика на основі активності в системах контролю версій [26]: коміти, pull requests, час злиття, code review activity. Використання цих метрик для оцінки продуктивності контроверсійне –

кількість комітів або рядків коду не корелює з якістю та може стимулювати негативну поведінку [27].

LinearB – платформа для аналізу продуктивності команд [28]. Інтегрується з Git, Jira, Slack. Фокусується на DORA metrics: Cycle Time, Deployment Frequency, Change Failure Rate, Mean Time to Recovery. Ці метрики орієнтовані на командний рівень та процеси, а не на індивідуальну продуктивність.

Jira/Tempo трекінг часу в контексті завдань [29]. Аналіз velocity команди, час на типи завдань, порівняння оцінки та фактичного часу. Вимагає дисципліни у логуванні, не надає інсайтів про якість використання часу.

Проблеми існуючих систем:

Інтеграція трекінгу часу з системою управління завданнями є логічною та зручною, проте вимагає дисципліни від розробників у регулярному логуванні часу. Крім того, Jira не надає інсайтів про якість використання часу – лише кількісні показники. Концептуальна схема інтеграції інструментів наведена на рисунку 1.2.

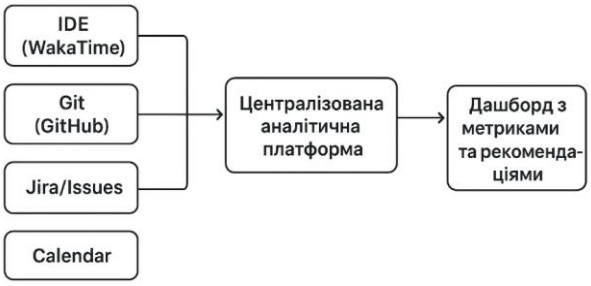

Рис. 1.2. – Концептуальна схема інтеграції різних інструментів для комплексного аналізу продуктивності

Проблеми існуючих систем аналізу:

1. **Відсутність адаптивності –** універсальні метрики без врахування індивідуальних особливостей, стилю роботи та персональних паттернів.
2. **Фокус на кількості –** метрики не корелюють з реальною цінністю роботи.
3. **Відсутність контексту переривань –** не аналізується вплив багатозадачності та перемикання контексту.

4. **Пасивний характер –** лише збір даних без проактивних рекомендацій чи нагадувань.
5. **Відсутність персоналізації –** універсальні рекомендації без врахування індивідуальних ритмів.
6. **Етичні питання –** інструменти сприймаються як інвазивний контроль, що знижує довіру.

Таким чином, існує потреба у комплексному рішенні, яке б поєднувало автоматичний збір даних, інтелектуальний аналіз паттернів продуктивності, адаптацію до індивідуальних особливостей та проактивну підтримку через персоналізовані рекомендації. Це становить основу для розробки системи, що пропонується в даній роботі.

1.3. Аналіз метрик ефективності роботи розробників ПЗ

Оцінка ефективності роботи розробників є складним завданням через багатовимірну природу діяльності. Розробка ПЗ вимагає врахування якісних аспектів, креативності, складності завдань та довгострокового впливу рішень.

1.3.1. Швидкість виконання завдань

Швидкість є базовою метрикою продуктивності, проте її інтерпретація вимагає обережності та контекстуального розуміння.

Velocity у Agile. У Scrum velocity визначається як кількість story points, виконаних командою за спринт [30].

Формула*: Velocity = Σ(Story Points завершених завдань) / Кількість спринтів.*

Дослідження 58 команд показало, що стабільна velocity (варіація <15%) корелює з передбачуваністю термінів та задоволеністю команди [31]. Використання velocity для порівняння команд або розробників некоректне через суб'єктивність оцінки story points.

Дослідження 58 команд розробки показало, що стабільна velocity (з варіацією менше 15% між спринтами) корелює з передбачуваністю термінів проєкту та задоволеністю команди [31]. Проте використання velocity для

порівняння різних команд або індивідуальних розробників є некоректним, оскільки оцінка story points є суб'єктивною та контекстно залежною.

Cycle Time та Lead Time. Cycle Time – час від початку роботи до завершення, Lead Time включає час очікування в черзі [32]. Рисунок 1.3 показує різницю між ними.

Рис. 1.3 – Різниця між Lead Time та Cycle Time

Систематичний моніторинг часу, витраченого на різні етапи розробки, дозволяє виявляти "вузькі місця" в процесі та покращувати загальну якість ПЗ. Дослідження [60] підкреслює взаємозв'язок між контролем якості на ранніх етапах розробки та загальною продуктивністю команди.

Аналіз даних 150 проєктів показав, що медіанний Cycle Time для типових завдань розробки становить 3.2 дні для junior розробників, 2.1 дня для middle та 1.4 дня для senior розробників [33].

Throughput. Вимірює кількість завершених завдань за одиницю часу. Формула: Throughput = Кількість завершених завдань / Період часу. Середній throughput: 6-8 завдань на тиждень для підтримки, 2-4 для нових фічей [34].

Таблиця 1.3 Метрики швидкості виконання завдань

Метрика	**Одиниця виміру**	**Типові значення**	**Переваги**	**Недоліки**
Velocity	Story Points/Sprint	20-40 SP	Враховує складність	Суб'єктивність оцінки

Продовження таблиці 1.3

Метрика	**Одиниця виміру**	**Типові значення**	**Переваги**	**Недоліки**
Cycle Time	Дні/Години	1-5 днів	Об'єктивність	Не враховує складність
Lead Time	Дні/Тижні	3-10 днів	Показує всі затримки	Включає час очікування
Throughput	Завдання/Тиждень	4-8 завдань	Простота розрахунку	Не враховує розмір завдань

Важливо відзначити, що жодна з цих метрик не є достатньою для повної оцінки продуктивності. Розробник може мати високий throughput, але низьку якість коду, що призведе до технічного боргу. Або навпаки – низький throughput через роботу над архітектурно складними рішеннями, що мають високу довгострокову цінність. Тому використовувати лише одну метрику для вимірювання продуктивності є недоречним рішенням. Краще використовувати комбінацію із декількох метрик.

1.3.2. Вплив переривань та відволікань на продуктивність

Переривання робочого процесу є одним з найбільш руйнівних факторів для продуктивності розробників. Численні дослідження підтверджують значний негативний вплив переривань на швидкість виконання завдань, якість роботи та рівень стресу.

Час відновлення після переривання. Дослідження Глорії Марк: після переривання потрібно в середньому 23 хвилини для відновлення концентрації [35]. Уточнені дані за типами:

- Короткі повідомлення: 8-12 хвилин.
- Незаплановані зустрічі: 20-25 хвилин.
- Технічні проблеми: 15-18 хвилин.
- Допомога колезі: 25-30 хвилин.

Вартість переривань. При 10-12 перериваннях на день сукупна втрата становить 3-4 години (40-50% робочого дня), що значною мірою спливає на продуктивність розробників. [36].

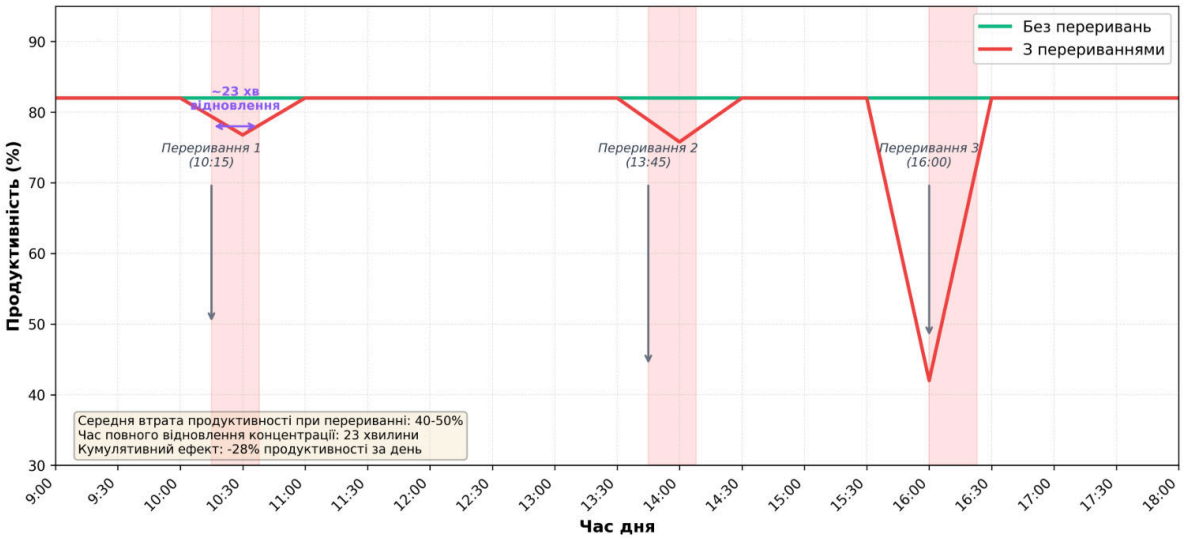

Рис. 1.4 – Вплив переривань на продуктивність протягом дня

Flow State. Концепція Міхая Чиксентміхаї – стан повної концентрації з продуктивністю в 5-10 разів вищою ніж зазвичай [37]. Для входження потрібно 15-30 хвилин безперервної концентрації на будь якому поставленому завданні перед людиною не залежно від рівня складності [38]. Будь-яке переривання негайно виводить з цього стану. Для повернення в цей стан потрібно все починати з початку. Час для входження залишається без змін.

Focus Score. Метрика для кількісної оцінки концентрації:

Focus Score = (Час безперервної роботи / Загальний робочий час) × 100%.

Дослідження 200 розробників: середній Focus Score 45-55%, високопродуктивні досягають 65-75% [39]. Показник <40% вказує на критичну проблему.

1.3.3. Багатозадачність та її вплив на якість коду

Людський мозок не здатний до справжньої багатозадачності – відбувається послідовне перемикання між завданнями.

Когнітивна вартість. Перемикання вимагає багато ресурсів для збереження/завантаження контексту та пригадування деталей завдання, яке стоїть перед розробником. Дослідження: перемикання між двома завданнями знижує продуктивність на 20%, трьома – на 40%, чотирма і більше – на 60-80% [40].

Вплив на якість коду. Експеримент з 120 розробниками показав прямий зв'язок між паралельними завданнями та дефектами [41]. Таблиця 1.4: робота над трьома+ завданнями призводить до >2x збільшення дефектів. Час на code review також зростає.

Таблиця 1.4 Вплив багатозадачності на якість коду

Кількість паралельних завдань	**Середня кількість багів на 1000 рядків коду**	**Збільшення відносно baseline**	**Code review час (хв/100 рядків)**
1 завдання (baseline)	0.8	-	15
2 завдання	1.3	+62%	18
3 завдання	2.1	+162%	23
+4 завдання	3.5	+337%	31

Як видно з таблиці, робота над трьома та більше завданнями одночасно призводить до більш ніж дворазового збільшення кількості дефектів у виконаному завданні, час на code review також зростає, оскільки код стає менш читабельним та більш заплутаним.

Work In Progress (WIP) Limits. Концепція Kanban обмежує завдання в роботі [42]. Оптимальні ліміти: Junior – 1 завдання, Middle – 1-2, Senior – 2-3. Компанії з строгими WIP limits відзначають:

- Скорочення cycle time на 35-45%.
- Зменшення дефектів на 25-30%.
- Підвищення пропускної здатності на 20-25%.
- Покращення передбачуваності термінів.

1.3.4. Важливість перерв та відпочинку

Регулярні перерви критично важливі для підтримання продуктивності. Культура "хасла" в IT-компаніях призводить до нехтування перервами з негативними наслідками.

Ультрадіанні ритми. Людина природно працює циклами по 90-120 хвилин [43]. Ігнорування призводить до зниження когнітивних здібностей, збільшення помилок, накопичення втоми та зниження креативності під час виконання завдань будь якої складності.

Оптимальний режим:

- **Мікроперерви** (30-60 секунд кожні 20-30 хвилин) – відведення погляду, вправи для очей.
- **Короткі перерви** (5-10 хвилин кожні 60-90 хвилин) – відхід від місця, легка активність.
- **Довга перерва** (30-60 хвилин після 3-4 годин) – обід, прогулянка, релаксація**.**

Вплив перерв на продуктивність. Дослідження Microsoft Research: розробники з регулярними перервами мають на 15-20% вищу продуктивність [44]. Рисунок 1.5 показує порівняння.

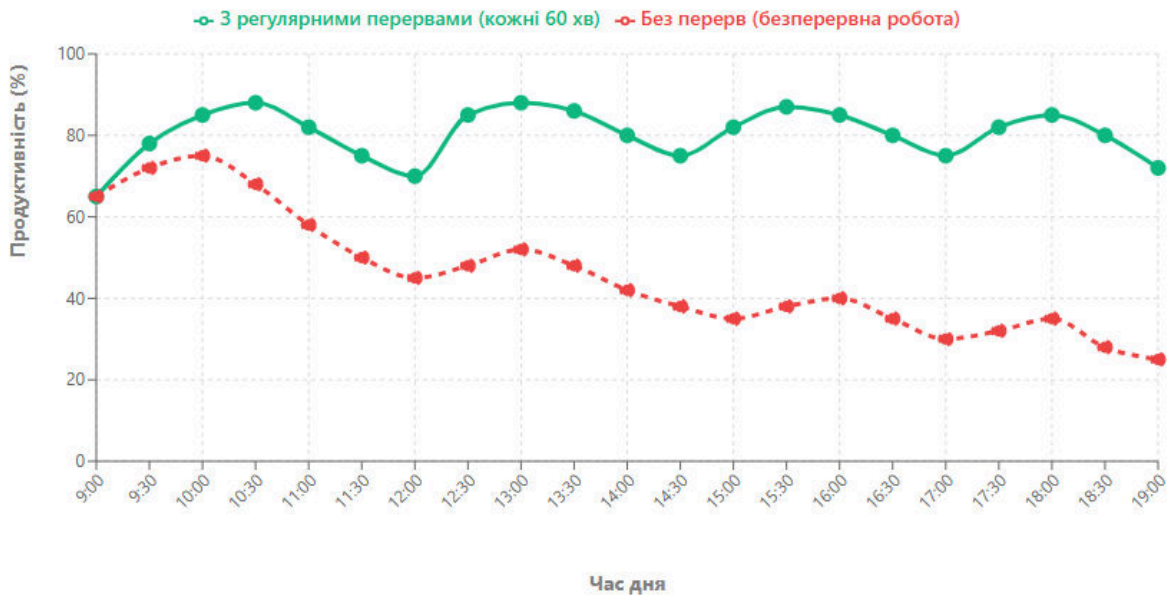

Рисунок 1.5 – Порівняння продуктивності з перервами та без

Типи відновлювальної активності [45]**.**

- **Високоефективні:** прогулянка (100%), медитація (95%), фізичні вправи (90%).
- **Середньоефективні:** соціальна взаємодія (70%), їжа (65%), читання (60%).
- **Низькоефективні:** соціальні мережі (30%), новини (25%), робоча пошта (15%).

Накопичувальний ефект втоми. Відсутність перерв призводить до накопичення втоми. Продуктивність у п'ятницю може бути на 30-40% нижчою порівняно з понеділком без правильного режиму відпочинку на протязі тижня [46].

Аналіз метрик показує, що продуктивність залежить не лише від навичок, але й від організації робочого процесу: мінімізації переривань, уникнення багатозадачності, дотримання регулярних перерв та усвідомленого управління увагою. Ці висновки формують основу для розробки системи оптимізації використання часу.

1.4. Адаптивні алгоритми в системах управління часом

Адаптивні системи, здатні навчатися на поведінці користувача та динамічно коригувати свою роботу, є перспективним напрямком розвитку інструментів управління продуктивністю [43]. На відміну від статичних систем з фіксованими правилами, адаптивні враховують індивідуальні особливості, звички, паттерни роботи та контекст діяльності.

Концепція адаптивності в управлінні часом. Адаптивний підхід базується на припущенні, що не існує універсального способу організації робочого процесу. Кожна людина має індивідуальні характеристики: хронотип (жайворонок/сова), оптимальну тривалість робочої сесії (25 хвилин – 3 години), різну чутливість до переривань, індивідуальні ритми продуктивності та різний баланс між швидкістю та якістю. Адаптивна система виявляє ці паттерни через аналіз історичних даних та коригує рекомендації відповідно.

Приклади адаптивних підходів. Концепція успішно застосовується в інших сферах:

- **Adaptive Learning Systems** (Coursera, Khan Academy) аналізують швидкість засвоєння та час на завдання для персоналізації темпу навчання [47].
- **Фітнес-трекери** (Garmin, Fitbit) аналізують історію тренувань, пульс, сон та стрес для персоналізованих рекомендацій [48].
- **Рекомендаційні системи** (Netflix, Spotify) використовують машинне навчання для аналізу вподобань [49].

Базові принципи адаптивних алгоритмів:

1. **Збір даних** – автоматичний збір про роботу розробників: час початку/завершення завдань, тривалість сесій, переривання, типи завдань, час доби, день тижня.
2. **Виявлення паттернів** – статистичний аналіз для виявлення закономірностей: коли найбільш продуктивний, оптимальна тривалість сесії, частота перерв, точність оцінки часу.
3. **Персоналізація рекомендацій** – генерація індивідуальних рекомендацій: оптимальний розклад з урахуванням піків продуктивності, реалістичні оцінки часу, персоналізовані інтервали для сповіщень, рекомендації щодо кількості завдань.
4. **Безперервне навчання** – постійне оновлення моделей на основі нових даних, адаптація до змін у звичках.

Математична основа. Простий адаптивний алгоритм для оцінки часу виконання:

$$Оцінка(t+1) = \alpha \times ФактичнийЧас(t) + (1 - \alpha) \times Оцінка(t)$$

де α (0.2-0.3) – коефіцієнт навчання. Для виявлення оптимального часу:

$$Продуктивність(час, тип\_завдання) = Виконано\_завдань / Витрачений\_час$$

Виклики:

- **Холодний старт** – недостатньо даних на початку, потрібні розумні значення за замовчуванням.

- **Шум у даних** – необхідно фільтрувати випадкові флуктуації від справжніх змін.
- **Інтерпретовність** – користувачі повинні розуміти рекомендації, які надає застосунок.
- **Приватність даних** – зберігання локально або з шифруванням.
- **Уникнення хибних оптимізацій** – не оптимізувати метрики, які легко "зламати".

Адаптивні підходи є перспективним напрямком, оскільки дозволяють подолати обмеження універсальних рішень та надати персоналізований досвід.

1.5. Обґрунтування необхідності розробки нового рішення

Проведений аналіз дозволяє сформулювати ключові недоліки поточного стану та обґрунтувати необхідність нового рішення.

Фрагментація інструментів. Різні аспекти покриваються різними інструментами, які погано інтегруються: WakaTime для IDE, Toggl для обліку часу, Jira для завдань, RescueTime для аналізу відволікань, окремі додатки для нагадувань. Кожен працює ізольовано, створюючи когнітивне навантаження та перешкоджаючи цілісній картині.

Відсутність адаптивності. Більшість інструментів використовують універсальний підхід. RescueTime має фіксовані категорії, Pomodoro – жорсткі 25-хвилинні інтервали, Jira надає однакові метрики всім. Оптимальна тривалість робочої сесії варіюється від 25 до 180 хвилин [50], ігнорування цих відмінностей призводить до субоптимальних рекомендацій.

Пасивний характер інструментів. Системи збирають та відображають дані, але не втручаються активно в процес. Розробник сам повинен переглядати статистику та змінювати свою поведінку в залежності від отриманих рекомендацій, що вимагає високої самодисципліни. Проактивна система з своєчасними нагадуваннями, попередженнями про перевантаження та рекомендаціями може бути значно ефективнішою.

Обмежений контекст аналізу. Інструменти фіксують лише кількісні показники, не аналізуючи якісні аспекти: відповідність складності завдання оцінці, якість коду, фактори впливу на продуктивність (атмосфера, обладнання, якість сну, досвід, фінансові стимули, згуртованість та стосунки в колективі), кореляцію між перервами та якістю роботи.

Недостатність експериментальних досліджень. Більшість рекомендацій базуються на анекдотичних свідченнях, а не на систематичних експериментах з розробниками ПЗ. Недостатньо досліджені питання: вплив перегляду статистики на мотивацію, оптимальна частота перерв для програмування, вплив планування на виконання, ефект типів сповіщень.

Таблиця 1.5 Зведення недоліків існуючих рішень

Аспект	**Існуючі рішення**	**Необхідне удосконалення**
Інтеграція	Фрагментовані інструменти	Єдина платформа
Персоналізація	Універсальні налаштування	Адаптація до індивідуальних паттернів
Проактивність	Пасивні звіти	Активні рекомендації та нагадування
Контекст	Кількісні метрики	Якісний аналіз з урахуванням факторів
Наукова обґрунтованість	Анекдотичні свідчення	Систематичні експерименти
Специфічність	Загальні інструменти	Орієнтація на розробників ПЗ

Вимоги до нового рішення:

- **Комплексність** – інтеграція управління завданнями, трекінгу часу, аналітики та нагадувань
- **Адаптивність** – навчання на поведінці користувача
- **Проактивність** – активна підтримка через своєчасні сповіщення
- **Контекстуальність** – аналіз кількісних та якісних аспектів

- **Науковість** – базування на експериментальних дослідженнях
- **Прозорість** – зрозумілі пояснення алгоритмів
- **Приватність** – повний контроль над даними
- **Простота використання** – мінімальні зусилля

Ці вимоги формують основу для проєктування системи, яка має подолати обмеження існуючих рішень через поєднання комплексного підходу, адаптивних алгоритмів та експериментально підтверджених практик.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз проблематики ефективного управління часом у розробці ПЗ та огляд існуючих підходів.
Встановлено, що неефективне управління часом є критичною проблемою: 68% розробників відзначають проблеми з продуктивністю, економічні втрати становлять сотні мільярдів доларів щорічно. Основні фактори: переривання (23 хвилини на відновлення), багатозадачність (зниження продуктивності до 60-80%), відсутність систематичного планування.

Проаналізовано чотири методології (Pomodoro, Time Boxing, GTD, Deep Work) та шість категорій інструментів (RescueTime, Toggl, WakaTime, Clockify, GitHub Insights, LinearB). Жодне рішення не забезпечує комплексного підходу та не враховує індивідуальні особливості.

Досліджено ключові метрики: швидкість виконання (Velocity, Cycle Time, Throughput), вплив переривань (Focus Score), наслідки багатозадачності (збільшення дефектів на 337% при 4+ завданнях), важливість перерв .

Розглянуто адаптивні алгоритми, які здатні навчатися на індивідуальних паттернах та надавати персоналізовані рекомендації.
Обґрунтовано необхідність нового рішення, яке поєднує комплексність, адаптивність, проактивність та базується на експериментальних дослідженнях. Сформульовано вісім ключових вимог до системи для подальшого проєктування в розділі 2.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ АДАПТИВНОГО АЛГОРИТМУ ТА АРХІТЕКТУРИ ЗАСТОСУНКУ

2.1 Функціональні та нефункціональні вимоги до системи

Проєктування системи управління часом та аналізу продуктивності розробників ПЗ починається з визначення вимог. Функціональні вимоги описують, що система повинна робити, нефункціональні – як система це робить з точки зору якості, продуктивності та безпеки.

2.1.1. Функціональні вимоги

Вимоги сформульовано відповідно до методології FURPS та пронумеровано для відстеження в процесі розробки.

FR-1. Управління обліковим записом користувача

- Реєстрація через email та пароль, автентифікація.
- Налаштування профілю (ім'я, фото, часовий пояс).
- Зміна паролю та відновлення доступу.

FR-2. Управління завданнями

- Створення завдань з обов'язковим заголовком та опціональним описом.
- Для кожного завдання: пріоритет (низький, середній, високий, критичний), дедлайн, оцінений час, теги.
- Підзавдання з вкладеністю до 2 рівнів.

- Редагування, видалення, архівування завдань.
- Фільтрація за статусом, пріоритетом, тегами, дедлайном.
- Сортування за датою створення, дедлайном, пріоритетом або власним порядком (drag-and-drop).

FR-3. Трекінг часу

- Старт/призупинення/відновлення/зупинка трекінгу одним кліком.
- Відображення активного таймера з часом.
- Автоматична фіксація переривань при відсутності взаємодії >5 хвилин.
- Підтвердження або відкидання часу переривання.
- Мануальне додавання записів часу для минулих періодів.
- Збереження історії всіх сесій з початком, завершенням та тривалістю.

FR-4. Планування робочого дня

- Створення плану дня з вибору завдань зі списку.
- Відображення загального оцінного часу та порівняння з доступним часом.
- Попередження про перевантаження при перевищенні на 20%+.
- Зміна порядку завдань drag-and-drop.

FR-5. Система сповіщень

- Сповіщення про наближення дедлайну (за 1 день, 3 години, 30 хвилин).
- Нагадування про перерви після тривалої роботи (за замовчуванням 60 хвилин, налаштовується).
- Ненав'язливі сповіщення з інформацією про тривалість перерви.
- Можливість відкласти на 10 хвилин або пропустити.
- Підсумкове сповіщення наприкінці дня з основними метриками.
- Налаштування типів бажаних сповіщень.

FR-6. Аналітика та візуалізація

- Дашборд з ключовими метриками за день: витрачений час, завершені завдання, Focus Score, переривання.
- Статистика за період (день, тиждень, місяць, діапазон).

- Графіки динаміки: витрачений час, velocity, тренд Focus Score.
- Розподіл часу за категоріями/тегами (кругова діаграма).
- Аналіз точності оцінок (estimated vs actual).
- Теплова карта продуктивності за часом доби та днями тижня.
- Тижневий звіт з підсумками та рекомендаціями.

FR-7. Адаптивні рекомендації

- Аналіз історичних даних для виявлення паттернів продуктивності.
- Корекція оцінок часу для нових завдань схожого типу.
- Рекомендації оптимального часу для різних типів завдань.
- Персоналізація інтервалів нагадувань про перерви.

FR-8. Експорт та звітність

- Експорт даних у форматах CSV та Excel.
- Генерація PDF-звіту з візуалізаціями та метриками.
- Деталізація за завданнями, категоріями та періодами.

Таблиця 2.1 Пріоритезація функціональних вимог

ID вимоги	**Назва функціональності**	**Пріоритет**	**Складність**	**Версія впровадження**
FR-1	Управління користувачем	Високий	Середня	MVP (v1.0)
FR-2	Управління завданнями	Високий	Висока	MVP (v1.0)
FR-3	Трекінг часу	Високий	Висока	MVP (v1.0)
FR-4	Планування дня	Середній	Середня	MVP (v1.0)
FR-5	Система сповіщень	Високий	Середня	MVP (v1.0)
FR-6	Аналітика базова	Високий	Висока	MVP (v1.0)
FR-6	Аналітика розширена	Середній	Середня	v1.1
FR-7	Адаптивні рекомендації	Середній	Висока	v1.1
FR-8	Експорт даних	Низький	Низька	v1.2

У рамках даної роботи реалізовано функціональність, позначену як MVP (Minimum Viable Product), яка включає всі критично необхідні можливості для проведення експериментального дослідження.

2.1.2. Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи та забезпечують прийнятний рівень продуктивності, надійності, безпеки та зручності.

NFR-1. Продуктивність

- Час відгуку на дії користувача ≤1 секунди.
- Завантаження головної сторінки ≤3 секунди.
- Генерація звітів за рік ≤5 секунд.
- Підтримка до 1000 одночасних користувачів без деградації.
- Ефективна обробка записів завдань та записів часових сесій.

NFR-2. Надійність та доступність

- Доступність 99.5% (≤3.6 годин downtime/місяць).
- Автозбереження кожні 30 секунд.
- Локальне збереження при втраті з'єднання та синхронізація після відновлення.
- Механізм відновлення після збоїв з автоперезапуском.
- Щоденне резервне копіювання з можливістю відновлення за останні 30 днів.

NFR-3. Безпека

- Хешування паролів (bcrypt або аналог).
- Шифрування через HTTPS (TLS 1.3+).
- Захист від SQL injection, XSS, CSRF.
- Сесії з обмеженим часом життя (24 години) та автопродовженням.
- Логування подій безпеки (спроби входу, зміна паролю, доступ до даних).
- Можливість видалення даних (GDPR right to be forgotten).

NFR-4. Зручність використання (Usability)

- Інтуїтивний інтерфейс без навчання для базових операцій.
- Основні дії доступні максимум за 2 кліки.
- Зрозумілі повідомлення про помилки з підказками.

- Адаптивний інтерфейс для різних розмірів екранів (desktop, tablet).
- Підтримка темної теми.
- Відповідність WCAG 2.1 рівня AA.

NFR-5. Сумісність

- Підтримка Chrome (v90+), Firefox (v88+), Edge (v90+), Safari (v14+).
- Інтеграція з календарями через iCal.
- REST API з документацією для майбутніх інтеграцій.

NFR-6. Масштабованість

- Горизонтальне масштабування через додавання серверів.
- Підтримка sharding бази даних.
- Кешування статичних ресурсів.

NFR-7. Підтримуваність та розширюваність

- Відповідність принципам Clean Code та SOLID.
- Покриття тестами ≥70% для критичних модулів.
- Модульна архітектура для додавання функцій без значних змін.
- Версіонування API для зворотної сумісності.
- Актуальна документація коду.

NFR-8. Локалізація та інтернаціоналізація

- Підтримка української та англійської мов.
- Текстові рядки в окремих файлах локалізації.
- Формати дати, часу та чисел відповідно до локалі.

Таблиця 2.2 Матриця нефункціональних вимог та їх пріоритетів

Категорія	**Ключові метрики**	**Цільове значення**	**Пріоритет**	**Метод верифікації**
Продуктивність	Час відгуку	<1 сек	Високий	Load testing
Надійність	Uptime	99.5%	Високий	Моніторинг
Безпека	Шифрування, хешування	HTTPS, bcrypt	Критичний	Security audit
Зручність	Кліки до дії	≤2 кліки	Високий	Usability testing

Продовження таблиці 2.2

Категорія	**Ключові метрики**	**Цільове значення**	**Пріоритет**	**Метод верифікації**
Сумісність	Підтримка браузерів	4 основні	Середній	Cross-browser testing
Масштабованість	Кількість користувачів	1000 одночасно	Середній	Stress testing
Підтримуваність	Code coverage	≥70%	Високий	Automated tests
Локалізація	Кількість мов	2 (UA, EN)	Низький	Manual testing

Визначені вимоги формують повну специфікацію системи та є основою для наступних етапів: розробки математичної моделі адаптивного алгоритму, проєктування архітектури та структури бази даних, реалізації системи. Пріоритезація дозволяє зосередитися на критичних аспектах у рамках обмежених ресурсів та часу.

2.2. Математична модель адаптивного алгоритму

Математична модель є основою для реалізації системи аналізу продуктивності розробників. Модель включає формалізацію задачі, набір метрик для кількісної оцінки ефективності та механізм адаптації на історичних даних.

2.2.1. Формалізація задачі

Маємо множину завдань $T = \{t_1, t_2, ..., t_n\}$, де кожне завдання t_i характеризується: ID, title, description, priority ∈ {Low, Medium, High, Critical}, estimated_time, actual_time, deadline, status ∈ {New, InProgress, Completed, Archived}, tags, created_at, completed_at.

Для кожного завдання система збирає робочі сесії $S(t_i) = \{s_1, s_2, ..., s_m\}$, де кожна сесія s_j описується: start_time, end_time, duration = end_time - start_time, interruptions = $\{int_1, int_2, ..., int_k\}$.

Переривання int_k характеризується: start, end, duration = end - start.

Загальний фактичний час: $actual\_time(t_i) = \Sigma_j(duration(s_j) - \Sigma_k\ duration(int_k))$

Мета системи:

- Максимізація продуктивності через оптимальне розподілення часу.
- Підвищення точності оцінки часу через адаптацію на історичних даних.
- Персоналізація рекомендацій відповідно до індивідуальних паттернів.

Задача оптимізації: Productivity(u, t) → max, де u – користувач, t – період.

Обмеження:

- Доступний час обмежений: Σ_i actual_time(t_i) ≤ available_time.
- Завершення до дедлайну: completed_at(t_i) ≤ deadline(t_i).
- Паралельні завдання обмежені: |{t_i : status = InProgress}| ≤ WIP_limit.

2.2.2. Алгоритм розрахунку метрик продуктивності

Для перегляду алгоритму дивись рисунок 2.1

1. Velocity (Швидкість виконання)

$Velocity(period) = |\{t_i : status = Completed \wedge completed\_at \in period\}| / |period|$

Зважена velocity: $WeightedVelocity(period) = \Sigma_i\ weight(t_i) / |period|$

де weight(t_i): 1 (до 2 год), 2 (2-4 год), 3 (4-8 год), 5 (понад 8 год).

2. Focus Score (Показник концентрації)

$FocusScore(period) = \Sigma_j\ focused\_time(s_j) / \Sigma_j\ duration(s_j)$

де $focused\_time(s_j) = duration(s_j) - \Sigma_k\ duration(int_k)$, $де\ duration(int_k) > 5\ хв$

Нормалізований: $FocusScore\% = FocusScore \times 100\%$

3. Efficiency (Ефективність оцінки часу)

$Efficiency(t_i) = min(estimated\_time(t_i), actual\_time(t_i)) / max(estimated\_time(t_i), actual\_time(t_i))$

Середня за період:

$AvgEfficiency(period) = (1/n) \times \Sigma_i\ Efficiency(t_i)$

Значення: 1.0, 0.8-0.99, 0.6-0.79, <0.6.

4. Break Compliance (Дотримання перерв)

$BreakCompliance(period) = actual\_breaks / recommended\_breaks$

де recommended_breaks = ⌊work_time / break_interval⌋

Значення: 0.8-1.2, 0.6-0.8 або 1.2-1.5, <0.6, >1.5.

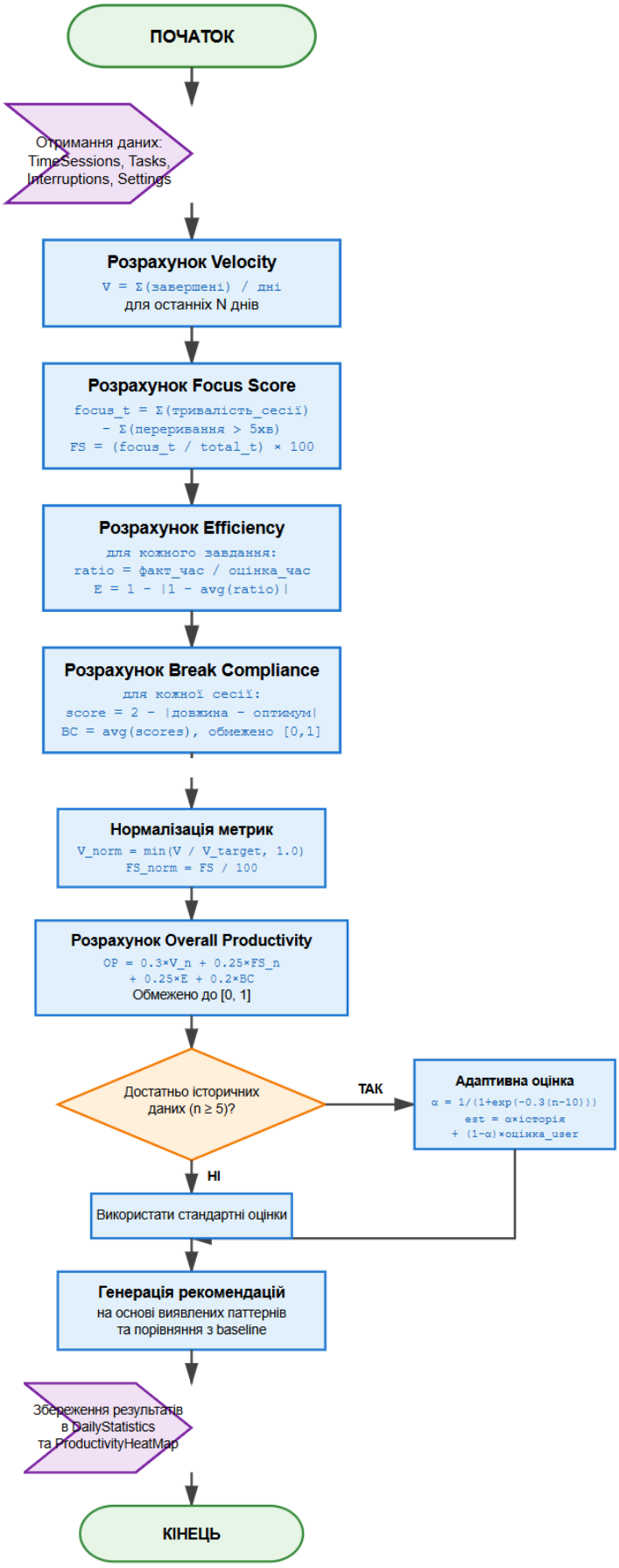

Рис 2.1 – Блок-схема алгоритму розрахунку метрик продуктивності

Композитна метрика продуктивності

$OverallProductivity = w_1 \times Velocity\_norm + w_2 \times FocusScore + w_3 \times Efficiency + w_4 \times BreakCompliance\_norm$

де $w_1 + w_2 + w_3 + w_4 = 1$, нормалізація:

$Velocity\_norm = Velocity(current) / Velocity(baseline)$

$BreakCompliance\_norm = 1 - |BreakCompliance - 1|$

За замовчуванням: $w_1 = 0.3$, $w_2 = 0.3$, $w_3 = 0.2$, $w_4 = 0.2$.

Таблиця 2.3 Метрики продуктивності та їх інтерпретація

Метрика	**Формула**	**Діапазон**	**Інтерпретація високих значень**
Velocity	Завдання/День	0-20+	Висока швидкість виконання
Focus Score	%	0-100%	Мало переривань, глибока концентрація
Efficiency	Ratio	0-1	Точне планування часу
Break Compliance	Ratio	0-2	Оптимальний режим відпочинку
Overall Productivity	Weighted Sum	0-1	Високий рівень продуктивності

2.2.3. Механізм адаптації на основі історичних даних

Адаптивний алгоритм включає три компоненти: адаптивну оцінку часу, персоналізацію інтервалів перерв та виявлення оптимальних часових вікон.

1. Адаптивна оцінка часу виконання

При створенні нового завдання система надає прогноз часу виконання на основі історичних даних про подібні завдання. Для цього використовується метод експоненційного згладжування з урахуванням категорії завдання:

$$EstimatedTime(t_{new}) = \alpha \times AvgActual(similar) + (1-\alpha) \times UserEstimate$$

де:

- $\alpha \in [0, 1]$ – коефіцієнт довіри до системи.
- $AvgActual(similar)$ – середній фактичний час для завершених завдань з такими ж тегами.
- $UserEstimate$ – початкова оцінка користувача (якщо є).

Для розрахунку α використовується логістична функція:

$$\alpha(n) = 1 / (1 + e^{-(n-10)/3})$$

де n – кількість завершених завдань у даній категорії. При n=0 α≈0 (повна довіра користувачу), при n=20 α≈0.95 (висока довіра системі).

Розрахунок AvgActual враховує тільки недавні завдання:

$AvgActual(category) = \Sigma_i\, w_i \times actual\_time(t_i) / \Sigma_i\, w_i$

де ваги визначаються експоненційним спаданням:

$w_i = e^{(-\lambda \times days\_ago(t_i))}$

$\lambda = 0.05$ забезпечує, що завдання тижневої давнини мають вагу ≈0.7, а місячної – ≈0.22.

2. Персоналізація інтервалів перерв

OptimalInterval = argmax(Correlation(interval, FocusScore)) interval∈[30, 120]

Система розбиває сесії на групи за тривалістю та розраховує середній Focus Score. Інтервал з найвищим Focus Score стає рекомендованим. Враховується динаміка втоми: коротші інтервали після обіду, довші вранці.

3. Виявлення оптимальних часових вікон

Виявлення оптимальних часових вікон

Теплова карта продуктивності: $HeatMap(hour, category) = (1/n) \times \Sigma_i\, Efficiency(t_i)$ де t_i – завдання категорії category, виконані в години hour. Приклад теплової карти продуктивності показано на рисунку 2.2.

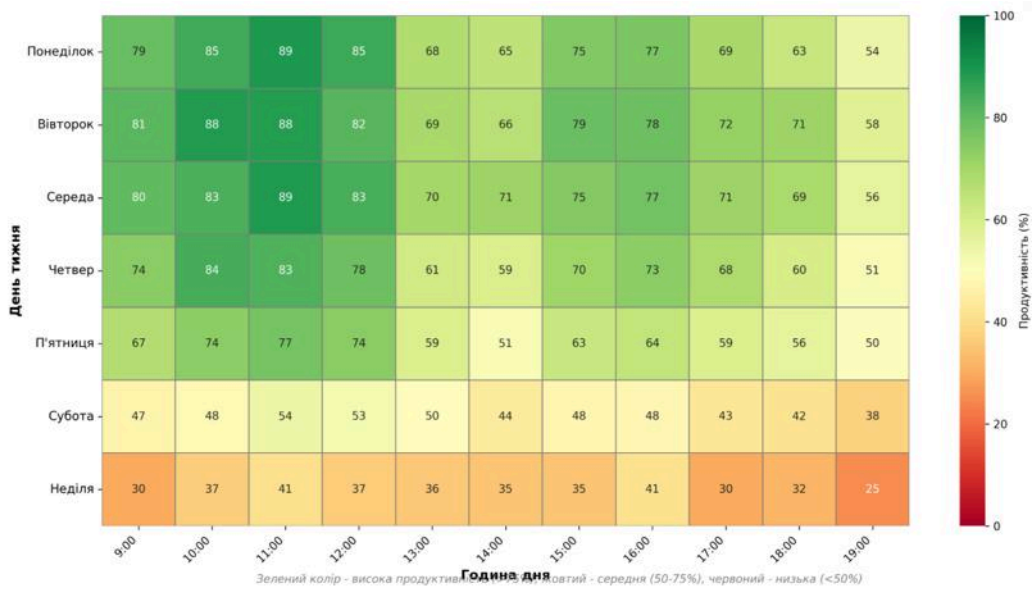

Рис. 2.2 – Приклад теплової карти продуктивності

На основі теплової карти система генерує рекомендації:

- "Найкращий час для складних завдань (coding): 8-11 ранку".
- "Рутинні завдання (documentation) краще виконувати після 15:00".
- "Ваша продуктивність знижується після обіду (12-13), рекомендуємо коротку прогулянку".

Таблиця 2.4 Параметри адаптивного алгоритму

Параметр	**Позначення**	**Діапазон**	**Значення за замовчуванням**	**Опис**
Коефіцієнт навчання	α	[0, 1]	Динамічний	Баланс між системою та користувачем
Коефіцієнт спадання	λ	[0, 1]	0.05	Швидкість "забування" старих даних
Вікно історії	window	30-180 днів	90 днів	Період для аналізу паттернів
Мінімум завдань	n_min	5-20	10	Мін. кількість для адаптації
Ваги композитної метрики	w_1, w_2, w_3, w_4	[0, 1]	0.3, 0.3, 0.2, 0.2	Важливість метрик

Оновлення моделі

Модель оновлюється автоматично при кожному завершенні завдання. Дивись рисунок 2.3

Граничні випадки

- Холодний старт (n=0): використання усереднених значень з досліджень.
- Аномалії: якщо фактичний час відрізняється >3x від прогнозу, завдання позначається як аномальне.
- Зміна контексту: автоматичне зменшення коефіцієнта довіри α при нових технологіях.

- Недостатність даних: для категорій з n<5 система показує низьку впевненість прогнозу

Математична модель забезпечує теоретичну основу для інтелектуальної системи, здатної персоналізувати роботу під індивідуальні особливості. Цикл навчання адаптивної моделі представлено на рисунку 2.3.

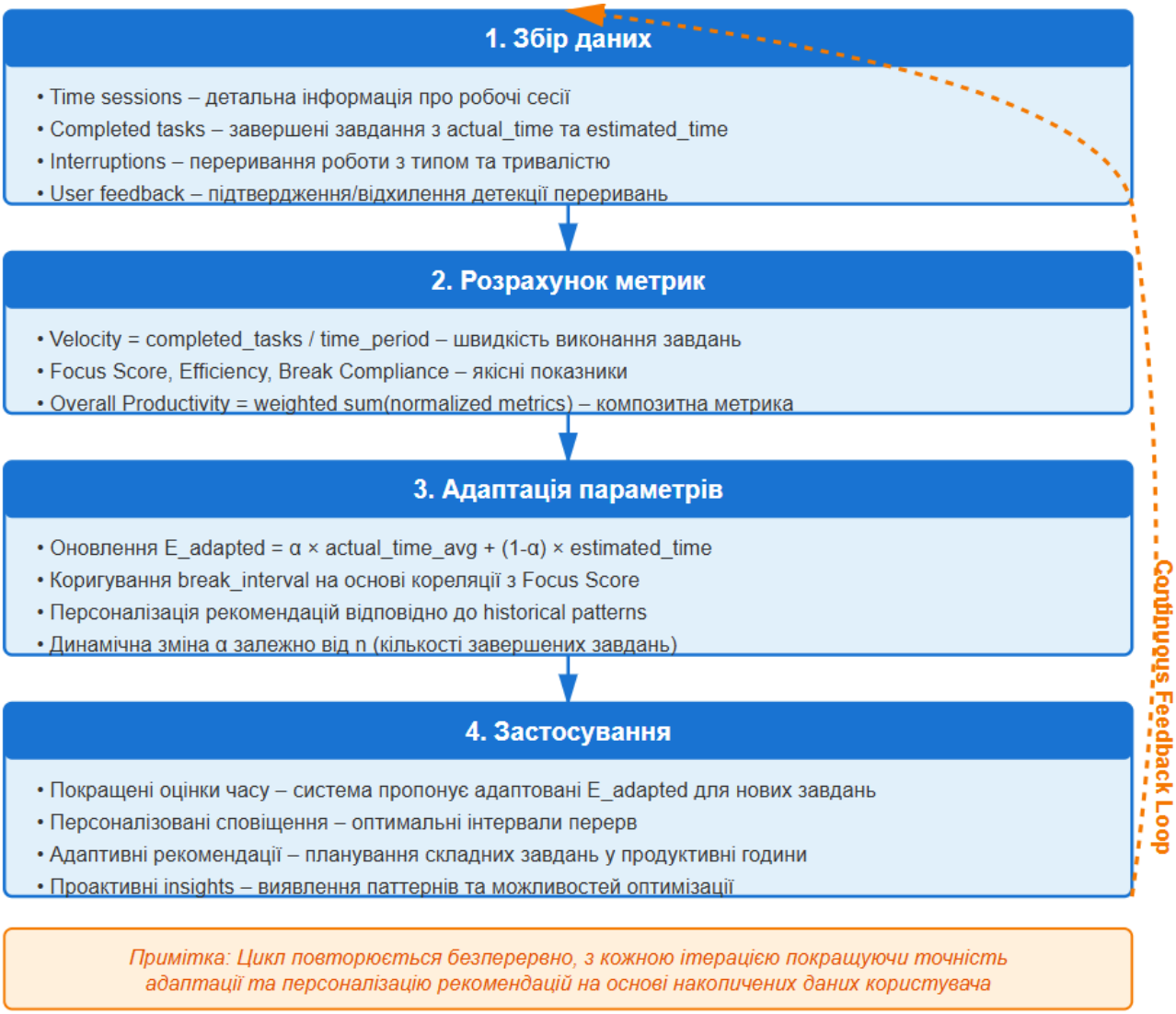

Рис. 2.3 – Цикл навчання адаптивної моделі

Математична модель забезпечує теоретичну основу для інтелектуальної системи, здатної персоналізувати роботу під індивідуальні особливості та безперервно покращувати точність прогнозів.

2.3. Архітектура системи

Архітектура програмної системи визначає її структуру, взаємодію компонентів, технологічний стек та підходи до організації коду і даних. При проєктуванні архітектури застосунку для аналізу ефективності часу розробників враховувались принципи модульності, масштабованості, підтримуваності та відповідності сучасним практикам веб-розробки.

2.3.1. Загальна архітектура системи

Система побудована за класичною клієнт-серверною. Високорівнева архітектура системи зображена на рисунку 2.4. Обрано архітектурний підхід на основі багатошарової (layered) архітектури, що забезпечує відокремлення бізнес-логіки від інфраструктурних деталей.

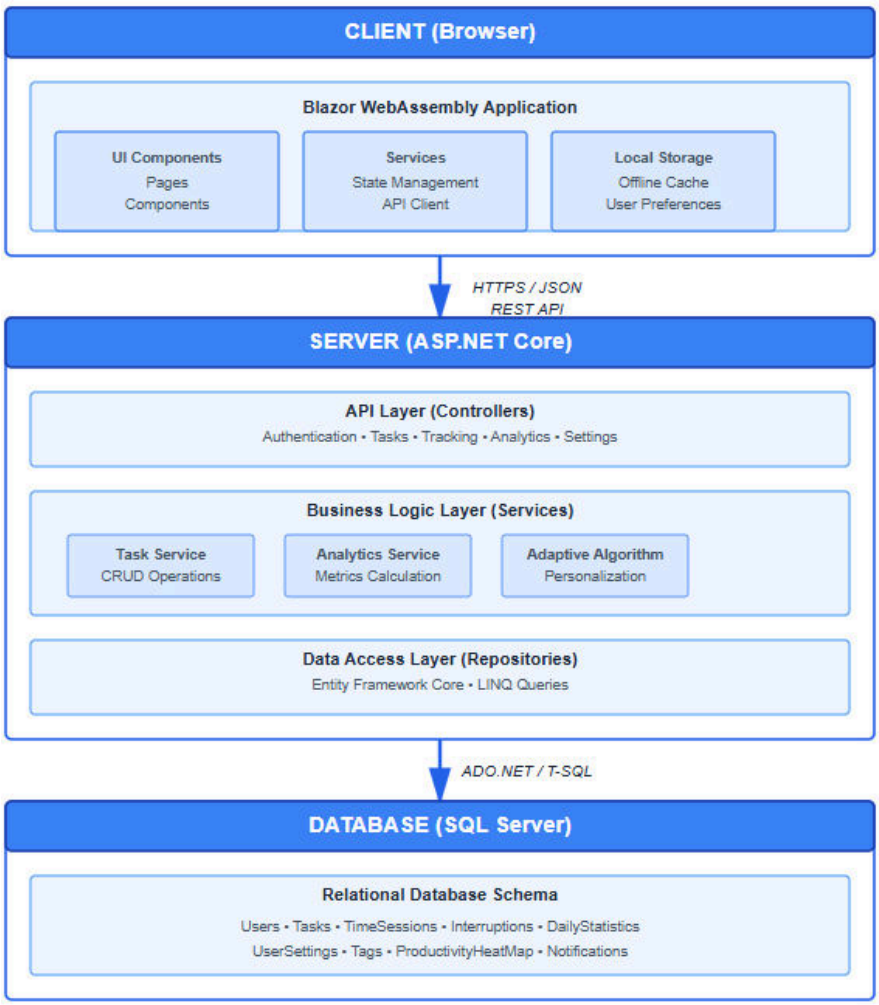

Рис. 2.4 – Високорівнева архітектура системи

Високорівнева архітектура

Система складається з трьох основних рівнів:

1. **Presentation Layer (Рівень представлення)** – клієнтський веб-застосунок, що забезпечує інтерактивний користувацький інтерфейс.
2. **Business Logic Layer (Рівень бізнес-логіки)** – серверний API на ASP.NET Core, що обробляє бізнес-логіку та забезпечує взаємодію з базою даних.
3. **Data Layer (Рівень даних)** – SQL Server база даних для збереження всіх даних системи з використанням Entity Framework Core як ORM для доступу до даних.

Ключові компоненти системи

1. Blazor WebAssembly Client

Клієнтська частина реалізована як Single Page Application (SPA) на Blazor WebAssembly, що дозволяє виконувати C# код безпосередньо в браузері користувача через WebAssembly. Основні переваги цього підходу:

- Єдина мова програмування (C#) для клієнта та сервера.
- Висока продуктивність через компіляцію в WebAssembly.
- Багата екосистема бібліотек .NET.
- Можливість повторного використання коду між клієнтом та сервером

Клієнт структурований за компонентним підходом:

- **Pages** – сторінки застосунку (Dashboard, Tasks, Analytics, Settings).
- **Components** – переиспользуемі UI компоненти (TaskCard, Timer, Chart).
- **Services** – клієнтські сервіси для управління станом та взаємодії з API.
- **Models** – моделі даних (DTOs для обміну з сервером).

2. ASP.NET Core Web API

Серверна частина побудована як RESTful API на ASP.NET Core 8.0. Архітектура API організована за принципами Clean Architecture:

- **Controllers** – приймають HTTP запити, валідують вхідні дані, викликають сервіси та повертають відповіді.
- **Services** – містять бізнес-логіку застосунку.
- **Repositories** – забезпечують доступ до даних через абстракції.

- **Models/Entities** – доменні сущності та DTOs.

Використовується DI для забезпечення слабкого зв'язування компонентів, полегшення тестування, гнучкості та простоті в майбутньому розширені.

3. Background Services

Для виконання фонових завдань (розрахунок статистики, відправка сповіщень, очищення старих даних) використовуються ASP.NET Core Hosted Services:

- **MetricsCalculationService** – періодично перераховує метрики продуктивності для активних користувачів.
- **NotificationService** – перевіряє умови для відправки.
- **DataCleanupService** – виконує очищення застарілих даних.

4. Entity Framework Core

Для роботи з базою даних використовується Entity Framework Core як ORM (Object-Relational Mapping). EF Core забезпечує:

- Автоматичну генерацію SQL запитів з LINQ виразів.
- Відстеження змін (change tracking) для оптимізації оновлень.
- Міграції для управління схемою БД.
- Підтримку транзакцій для забезпечення консистентності даних.

Патерни проєктування

При розробці архітектури застосовано наступні патерни:

1. **Repository Pattern** – абстрагування логіки доступу до даних, що дозволяє легко змінювати реалізацію БД без впливу на бізнес-логіку.
2. **Service Layer Pattern** – інкапсуляція бізнес-логіки в окремі сервіси, що робить код більш модульним та тестовним.
3. **DTO** – використання окремих об'єктів для передачі даних між клієнтом та сервером, що захищає внутрішню структуру доменних моделей.
4. **Dependency Injection** – всі залежності впроваджуються через конструктори, що забезпечує гнучкість та спрощує unit testing.
5. **Factory Pattern** – для створення складних об'єктів (наприклад, звітів з різною конфігурацією).

Таблиця 2.5 Технологічний стек системи

Компонент	**Технологія**	**Версія**	**Призначення**
Frontend Framework	Blazor WebAssembly	.NET 8.0	Клієнтський SPA застосунок
Backend Framework	ASP.NET Core	8.0	RESTful Web API
ORM	Entity Framework Core	8.0	Доступ до бази даних
Database	SQL Server	2022	Реляційна база даних
UI Components	MudBlazor	6.x	Готові UI компоненти
Charts	Chart.js (via Blazor)	4.x	Візуалізація графіків
Authentication	ASP.NET Core Identity	8.0	Аутентифікація та авторизація
HTTP Client	HttpClient	.NET 8.0	Комунікація клієнт-сервер
JSON Serialization	System.Text.Json	.NET 8.0	Серіалізація даних

Переваги обраної архітектури:

- Чітке розділення відповідальності між шарами.
- Висока тестованість завдяки DI та абстракціям.
- Можливість незалежної розробки frontend та backend.
- Легка заміна окремих компонентів без впливу на інші.

2.3.2. Структура бази даних

База даних спроєктована за третьою нормальною. ER-діаграма бази даних наведена на рисунку 2.5.

Опис таблиць бази даних

1. Users – зберігає інформацію про зареєстрованих користувачів.

2. UserSettings – персональні налаштування користувача.

3. Tasks – завдання користувача.

4. Tags – категорії/теги завдань.

5. TaskTags – зв'язок Many-to-Many між Tasks та Tags.

6. TimeSessions – сесії роботи над завданнями.

7. Interruptions – переривання під час роботи.

8. DailyStatistics – агреговані статистики по днях.

9. ProductivityHeatMap – дані для теплової карти продуктивності.

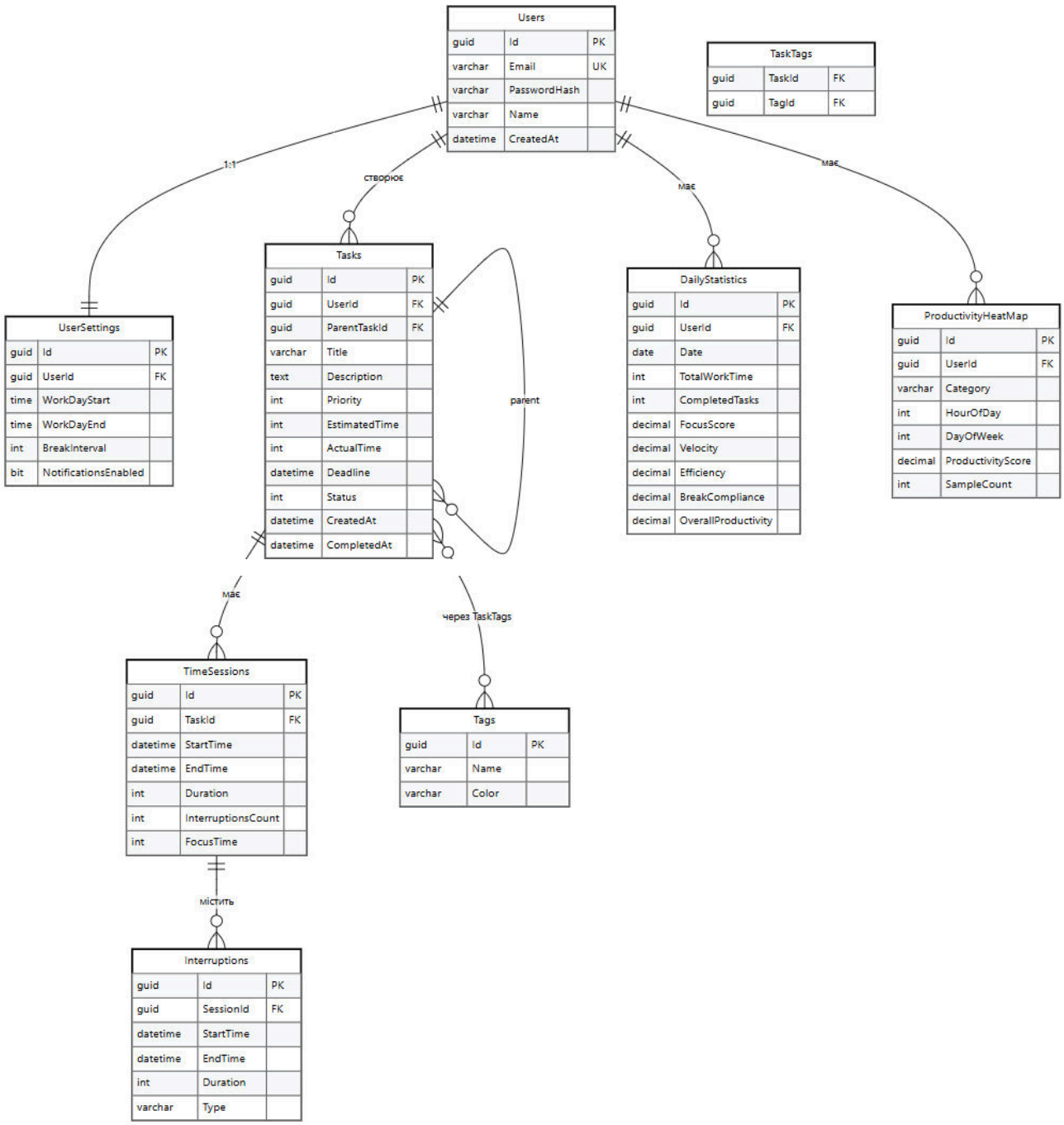

Рис. 2.5 – ER-діаграма бази даних

2.3.3. API та взаємодія компонентів

RESTful API є основним інтерфейсом взаємодії між клієнтською та серверною частинами системи. API спроєктовано відповідно до принципів REST та використовує JSON як формат обміну даними.

Структура API endpoints

API організовано за ресурсами (resource-oriented), кожен ресурс має стандартний набір операцій CRUD:

1. Authentication API – автентифікація та управління обліковим записом користувача.
2. Tasks API – управління завданнями.
3. Tracking API – трекінг часу.
4. Analytics API – аналітика та статистика.
5. Notifications API – управління сповіщеннями.
6. Settings API – налаштування користувача.
7. Reports API – генерація звітів.

Таблиця 2.6 Приклади API запитів та відповідей

Endpoint	**Method**	**Request Body**	**Response**	**Status Code**
/api/tasks	POST	{title, description, estimatedTime}	{id, ...}	201 Created
/api/tracking/start	POST	{taskId}	{sessionId, startTime}	200 OK
/api/analytics/daily	GET	Query: ?date=2024-11-15	{metrics...}	200 OK
/api/tasks/{id}	DELETE			204 No Content

Автентифікація та авторизація

Система використовує JWT для автентифікації користувача. Діаграму послідовності взаємодії компонентів застосунку продемонстровано на рисунку 2.6.

1. Користувач відправляє POST /api/auth/login з email та паролем.
2. Сервер перевіряє облікові дані та генерує JWT token.
3. Клієнт зберігає token в Local Storage.
4. При кожному запиті клієнт додає header: Authorization: Bearer {token}.
5. Сервер валідує token та ідентифікує користувача.

Структура JWT token: {"sub": "user_id", "email": "user@example.com", "exp": 1700000000, "iat": 1699900000 }

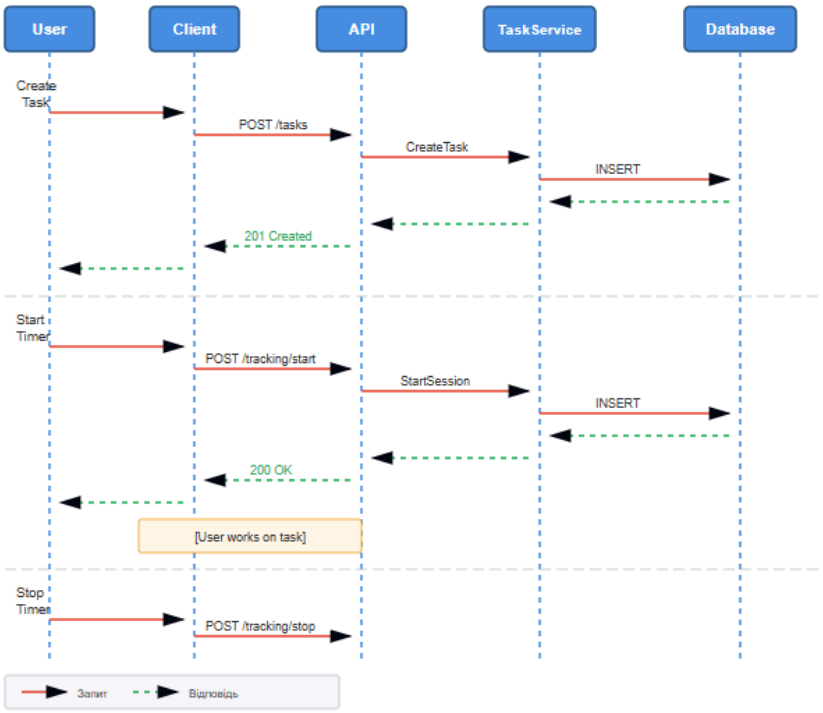

Рис. 2.6. – Діаграма послідовності: Створення та виконання завдання

Обробка помилок

API використовує стандартні HTTP статус коди для індикації результату:

- 200 OK – успішний запит.
- 201 Created – ресурс створено.
- 204 No Content – успішно, без даних у відповіді.
- 400 Bad Request – невалідні вхідні дані.
- 401 Unauthorized – відсутня або невалідна автентифікація.
- 403 Forbidden – немає доступу до ресурсу.
- 404 Not Found – ресурс не знайдено.
- 409 Conflict – конфлікт (наприклад, email вже існує).
- 500 Internal Server Error – помилка сервера.

Формат повідомлення про помилку:

{ "error": "ValidationError", "message": "Task title is required","details": { "field": "title", "constraint": "required" } }

Оптимізація продуктивності

Для підвищення продуктивності API застосовуються наступні техніки:

1. **Response Caching** – кешування відповідей для read-only endpoints.
2. **Pagination** – обмеження кількості записів у відповіді.
3. **Eager/Lazy Loading** – оптимізація завантаження зв'язаних сутностей в EF Core.
4. **Compression** – gzip стиснення відповідей для економії bandwidth.
5. **Connection Pooling** – переиспользование з'єднань з базою даних.

Спроєктована архітектура забезпечує модульність, масштабованість та підтримуваність системи, а чітко визначений API дозволяє незалежну розробку клієнтської та серверної частин і спрощує майбутню інтеграцію з зовнішніми системами.

2.4. Проєктування користувацького інтерфейсу

Користувацький інтерфейс є критично важливим елементом системи, оскільки безпосередньо впливає на досвід користувача та ефективність використання застосунку. При проєктуванні UI керувались принципами простоти, інтуїтивності та мінімалізму, щоб не відволікати користувача від основної роботи.

Принципи проєктування інтерфейсу

1. **Мінімалізм** – інтерфейс містить тільки необхідні елементи, уникаючи інформаційного перевантаження.
2. **Швидкий доступ до основних дій** – створення завдання та старт/стоп таймера доступні в 1-2 кліки з будь-якого екрану.
3. **Візуальна ієрархія** – використання розміру, кольору та розташування для виділення найважливішої інформації.
4. **Зворотний зв'язок** – система негайно реагує на дії користувача (анімації, повідомлення, зміна стану).
5. **Консистентність** – єдиний стиль кнопок, форм, кольорів та типографіки по всьому застосунку.

6. **Адаптивність** – інтерфейс коректно відображається на різних розмірах екрану (1920x1080, 1366x768, планшети).

Структура навігації

Застосунок має просту та зрозумілу структуру з п'яти основних розділів. Структура навігації застосунку представлена на рисунку 2.7.

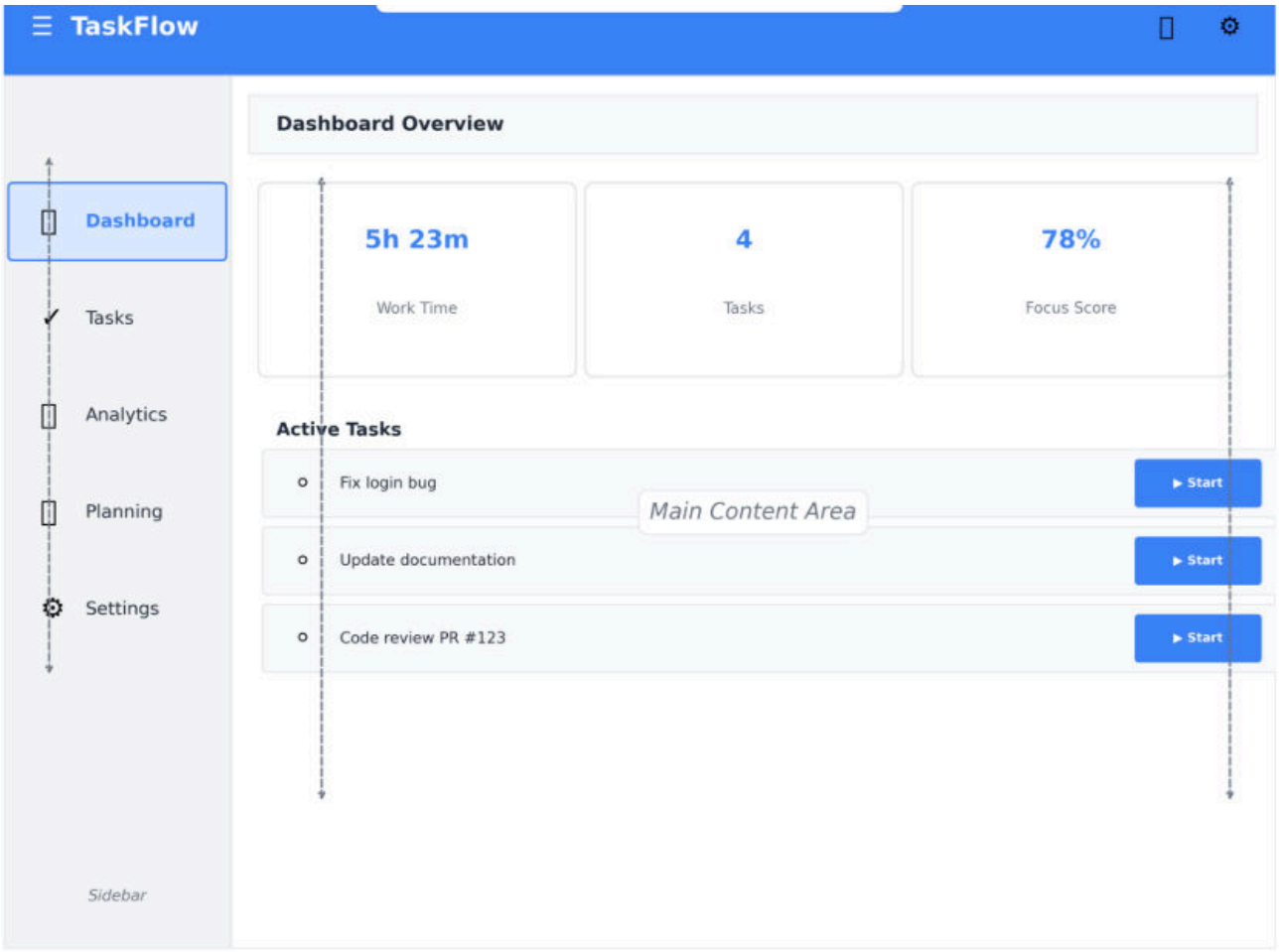

Рисунок 2.7 – Структура навігації застосунку

Опис основних екранів

1. Dashboard (Дашборд)

Головний екран застосунку, що надає швидкий огляд поточної продуктивності наведено на рисунку 2.8.

- **Active Timer** – великий помітний таймер з часом активної сесії.
- **Quick Actions** – кнопки швидких дій (Start Task, Create Task, Take Break).
- **Today's Metrics** – картки з метриками за день:

- Витрачений час
- Завершені завдання
- Focus Score
- Кількість переривань

- **Tasks List (Compact)** – скорочений список активних завдань (3-5).
- **Quick Stats Chart** – мініатюрний графік продуктивності за тиждень.

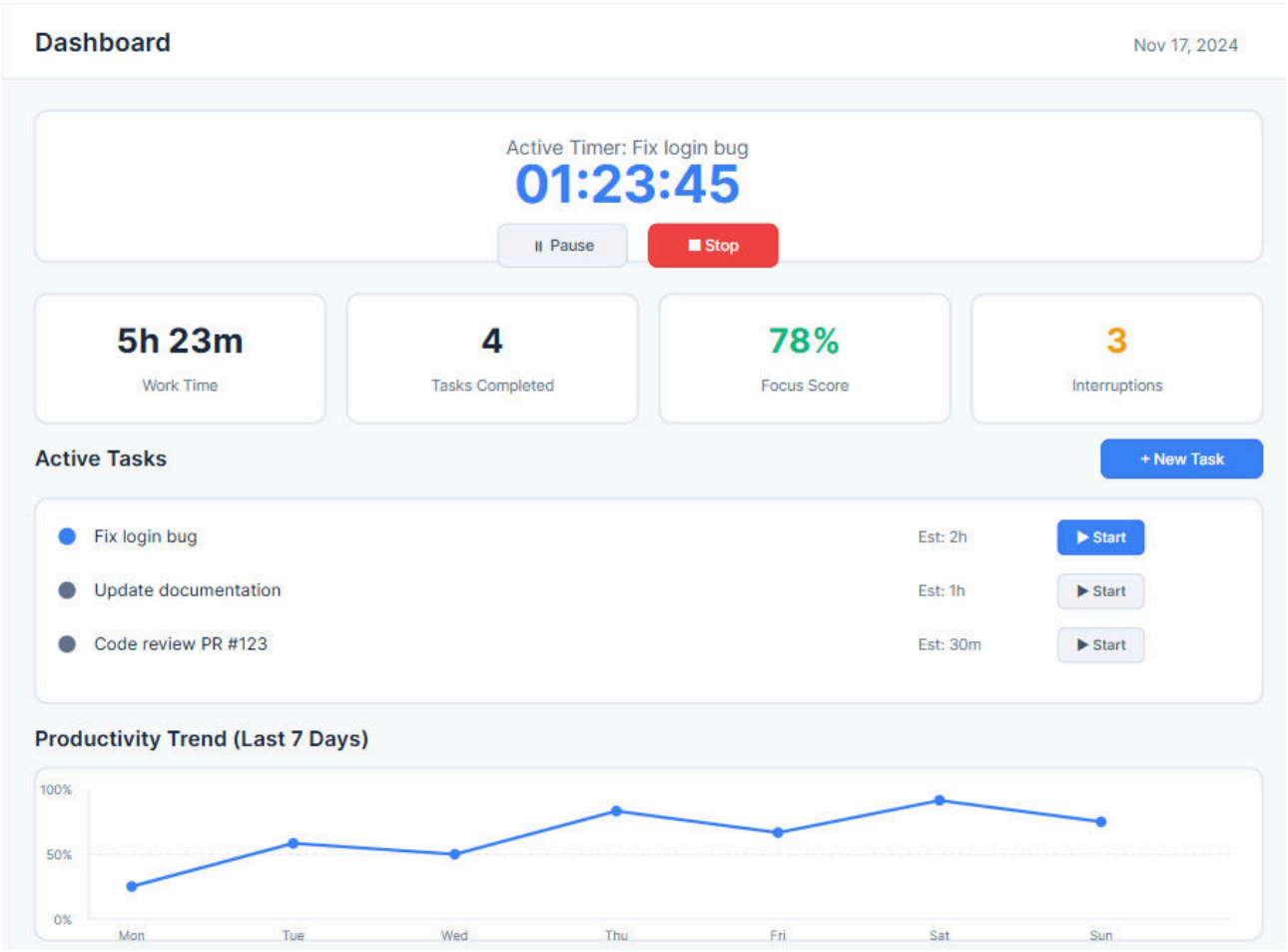

Рис. 2.8 – Макет екрану Dashboard

2. Tasks (Завдання)

Екран управління завданнями з можливістю фільтрації, сортування та групування:

- **Filters Bar** – фільтри за статусом, пріоритетом, тегами, дедлайном.
- **View Modes** – перемикання між видами (List, Board/Kanban).
- **Task Cards** – картки завдань з інформацією:

- Заголовок та опис.
- Пріоритет (кольоровий індикатор).
- Теги.
- Оцінений vs фактичний час.
- Дедлайн (з підсвічуванням якщо скоро).
- Кнопки дій (Start, Edit, Delete).

- **Create Task Form** – модальне вікно або бічна панель для створення.

3. Analytics (Аналітика)

Екран з детальною статистикою та візуалізаціями:

- **Period Selector** – вибір періоду (Day, Week, Month, Custom Range).
- **Key Metrics Cards** – великі картки з основними метриками.
- **Charts Section** – графіки:
  - Line Chart: Продуктивність по днях.
  - Bar Chart: Час за категоріями.
  - Pie Chart: Розподіл часу за проєктами.
  - Heat Map: Продуктивність за годинами та днями.
- **Trends & Insights** – текстові інсайти.
- **Export Button** – експорт звіту в PDF/CSV.

4. Planning (Планування)

Екран планування робочого дня:

- **Calendar View** – календар на тиждень або день.
- **Drag-and-Drop** – перетягування завдань на часові слоти.
- **Time Blocks** – візуалізація запланованого часу.
- **Workload Indicator** – індикатор завантаженості.
- **Recommendations** – рекомендації системи на основі аналізу.

5. Settings (Налаштування)

Екран налаштувань користувача:

- **Profile** – ім'я, email, фото.
- **Work Schedule** – робочі години, часовий пояс.
- **Notifications** – налаштування сповіщень (типи, інтервали).

- **Appearance** – темна/світла тема, мова інтерфейсу.
- **Privacy** – управління даними, експорт, видалення акаунту.

Кольорова палітра

Система використовує поміркована кольорову схему, що не відволікає та добре сприймається:

- **Primary Color** – #3B82F6 (синій) – основні дії, активні елементи.
- **Secondary Color** – #8B5CF6 (фіолетовий) – акценти.
- **Success** – #10B981 (зелений) – завершені завдання, позитивні метрики.
- **Warning** – #F59E0B (помаранчевий) – попередження, наближення дедлайнів.
- **Danger** – #EF4444 (червоний) – критичні дедлайни, помилки.
- **Neutral** – #6B7280 (сірий) – текст, рамки.

Типографіка

- **Font Family** – Inter (веб-шрифт від Google Fonts) – відмінна читабельність.
- **Headings** – 24px/32px/20px (H1/H2/H3), font-weight: 600.
- **Body Text** – 14px, font-weight: 400, line-height: 1.5.
- **Small Text** – 12px для міток та допоміжної інформації.

Адаптивність та доступність

- Breakpoints: Desktop (>1200px), Tablet (768-1199px).
- Контрастність тексту відповідає WCAG 2.1 Level AA (мінімум 4.5:1).
- Всі інтерактивні елементи мають розмір мінімум 44x44px для зручності.
- Підтримка навігації з клавіатури (Tab, Enter, Esc).

Спроєктований інтерфейс забезпечує баланс між функціональністю та простотою використання, дозволяючи користувачам швидко виконувати необхідні дії без зайвих кліків та відволікань.

2.5. Обрані технології та обґрунтування вибору

Вибір технологічного стеку є критично важливим рішенням, що впливає на продуктивність розробки, швидкодію системи, можливості масштабування та

підтримки. При виборі технологій керувались критеріями: продуктивність, екосистема, документація, підтримка спільноти, перспективність та особистий досвід.

Frontend: Blazor WebAssembly

Обґрунтування вибору:

Blazor WebAssembly обрано як основну технологію для клієнтської частини з наступних причин:

1. **Єдина мова програмування** – використання C# як на клієнті, так і на сервері застосунку дозволяє розробляти всю систему однією мовою, що спрощує розробку, зменшує context switching та дозволяє перевикористовувати код.
2. **Продуктивність** – після компіляції в WebAssembly код виконується близько до нативної швидкості, що забезпечує оптимальний відгук інтерфейсу.
3. **Екосистема .NET** – доступ до величезної кількості бібліотек NuGet, включаючи складні компоненти UI, графіки, робота з різними типами даних та інше.
4. **Type Safety** – строга типізація C# запобігає багатьом помилкам на етапі компіляції, на відміну від JavaScript.
5. **Відсутність необхідності Node.js** – вся розробка ведеться в екосистемі .NET без потреби підтримувати окреме JavaScript середовище.

Альтернативи та чому відхилені:

- **React/Vue.js** – потребує знання JavaScript/TypeScript, окремої екосистеми, складнішої інтеграції з .NET backend.
- **Angular** – складний для проєкту такого масштабу, steep learning curve
- **Blazor Server** – потребує постійного з'єднання з сервером.

Backend: ASP.NET Core Web API

Обґрунтування вибору:

1. **Високу продуктивність** – ASP.NET Core є одним з найшвидших веб-фреймворків випереджаючи Node.js, Django, Spring Boot.

2. **Cross-platform** – працює на Windows, Linux, macOS, що забезпечує гнучкість при деплої.
3. **Вбудований Dependency Injection** – нативна підтримка DI спрощує написання тестовного та модульного коду.
4. **Багата функціональність out-of-the-box** – middleware для автентифікації, CORS, логування, кешування, compression вже вбудовані.
5. **Entity Framework Core** – потужний ORM з підтримкою міграцій, LINQ запитів, відстеження змін.
6. **Активна підтримка Microsoft** – регулярні оновлення, довгострокова підтримка LTS версій, велика спільнота.

Альтернативи:

- **Node.js/Express** – асинхронність хороша для I/O, але слабша типізація та нижча продуктивність для CPU-intensive операцій.
- **Django/FastAPI** – не оптимальний для CPU-intensive обчислень метрик.
- **Spring Boot** – складніший для невеликих проєктів, довші цикли розробки.

Database: SQL Server

Обґрунтування вибору:

1. **Повна інтеграція з .NET** – EF Core має найкращу підтримку SQL Server.
2. **Транзакційна цілісність** – ACID властивості критичні для фінансового обліку часу та консистентності метрик.
3. **Потужні можливості запитів** – T-SQL для складних аналітичних запитів, window functions, CTEs.
4. **Performance** – індекси, query optimizer, execution plans для оптимізації.
5. **Інструменти** – SQL Server Management Studio (SSMS) для адміністрування та налагодження.

Альтернативи:

- **PostgreSQL** – відмінний вибір, але SQL Server обрано через кращу інтеграцію з .NET та наявний досвід.
- **MongoDB** – NoSQL не підходить для даних з чіткими зв'язками.

UI Framework: MudBlazor

Обґрунтування вибору:

MudBlazor обрано як бібліотеку UI компонентів:

1. **Material Design** – сучасний та знайомий дизайн.
2. **Повний набір компонентів** – tables, forms, dialogs, charts, datepickers.
3. **Відмінна документація** – детальні приклади та API reference.
4. **Active development** – регулярні оновлення та підтримка.
5. **Customization** – легко налаштовувати теми та стилі.

Charts: Chart.js через Blazor Wrapper

Для візуалізації графіків обрано Chart.js як найпопулярнішу та потужну бібліотеку з підтримкою всіх типів графіків (line, bar, pie, radar, scatter) та інтерактивності.

Таблиця 2.7 Порівняння технологічних альтернатив

Компонент	**Обрана технологія**	**Альтернатива 1**	**Альтернатива 2**	**Переваги обраної**
Frontend	Blazor WASM	React	Vue.js	Єдина мова, type safety, .NET ecosystem
Backend	ASP.NET Core	Node.js/Express	FastAPI (Python)	Продуктивність, DI, EF Core
Database	SQL Server	PostgreSQL	MongoDB	Інтеграція з .NET, транзакції, T-SQL
UI Library	MudBlazor	Radzen	Bootstrap	Material Design, документація
Charts	Chart.js	Plotly	Recharts	Популярність, функціональність

Обрані технології формують сучасний, продуктивний та підтримуваний стек, що дозволяє ефективно реалізувати всі функціональні вимоги системи та забезпечити хороший developer experience під час розробки.

Висновки до розділу 2

У другому розділі виконано повне проєктування системи управління часом та аналізу продуктивності розробників ПЗ.

Визначено 35 функціональних вимог, згрупованих у 8 категорій, та 8 груп нефункціональних вимог, що охоплюють продуктивність, надійність, безпеку, зручність використання, сумісність, масштабованість, підтримуваність та локалізацію.

Розроблено математичну модель адаптивного алгоритму, що включає формалізацію задачі з математичними позначеннями, чотири ключові метрики продуктивності з формулами для їх розрахунку, та механізм адаптації на основі експоненційного згладжування історичних даних з коефіцієнтом довіри α, що динамічно зростає з накопиченням статистики.

Спроєктовано багатошарову архітектуру системи з чітким розділенням на presentation, business logic та data layers. Створено структуру реляційної бази даних з 9 таблицями у третій нормальній формі з відповідними індексами та обмеженнями цілісності. Визначено 40+ RESTful API endpoints для взаємодії клієнта та сервера з використанням JWT для автентифікації.

Спроєктовано користувацький інтерфейс з п'ятьма основними екранами (Dashboard, Tasks, Analytics, Planning, Settings), що забезпечує швидкий доступ до основних функцій та відповідає принципам простоти, мінімалізму та консистентності. Визначено кольорову палітру, типографіку та макети екранів.

Обґрунтовано вибір технологічного стеку: Blazor WebAssembly для frontend, ASP.NET Core для backend, SQL Server для бази даних, MudBlazor для UI компонентів. Вибір базується на критеріях продуктивності, інтеграції, type safety та єдиної мови програмування для всього стеку.

Розроблене проєктне рішення формує повну технічну специфікацію для реалізації системи в розділі 3.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

3.1 Реалізація основних модулів застосунку

Реалізація системи виконана відповідно до спроєктованої архітектури з використанням обраного технологічного стеку. Розробка велась за ітеративним підходом з поступовою реалізацією функціональності від базових можливостей до складних адаптивних алгоритмів.

Структура проєкту

Проєкт організовано як Visual Studio Solution з трьома основними проєктами:

TaskFlow.Client (Blazor WebAssembly) – клієнтська частина застосунку, містить сторінки (Dashboard, Tasks, Analytics, Planning, Settings), переиспользувані компоненти (TaskCard, TimerWidget, MetricCard, ChartComponent, NotificationPanel), клієнтські сервіси (ApiClient, StateManager, LocalStorageService, NotificationService), DTOs та моделі (TaskDto, TimeSessionDto, StatisticsDto, UserSettingsDto), а також статичні ресурси (CSS, JS, index.html).

TaskFlow.Server (ASP.NET Core Web API) – серверна частина застосунку, включає API контролери (AuthController, TasksController, TrackingController, AnalyticsController, SettingsController), бізнес-логіку в сервісах (TaskService, TrackingService, AnalyticsService, NotificationService, AdaptiveAlgorithmService), фонові сервіси (MetricsCalculationService,

NotificationWorker), доступ до даних (ApplicationDbContext, Repositories, Migrations) та точку входу (Program.cs).

TaskFlow.Shared – спільна бібліотека з доменними моделями (User, Task, TimeSession, DailyStatistics), Data Transfer Objects та константами з енумами.

Архітектурні рішення

Реалізація базується на Clean Architecture з чітким розділенням залежностей. Структура залежностей між компонентами показана на рисунку 3.1.

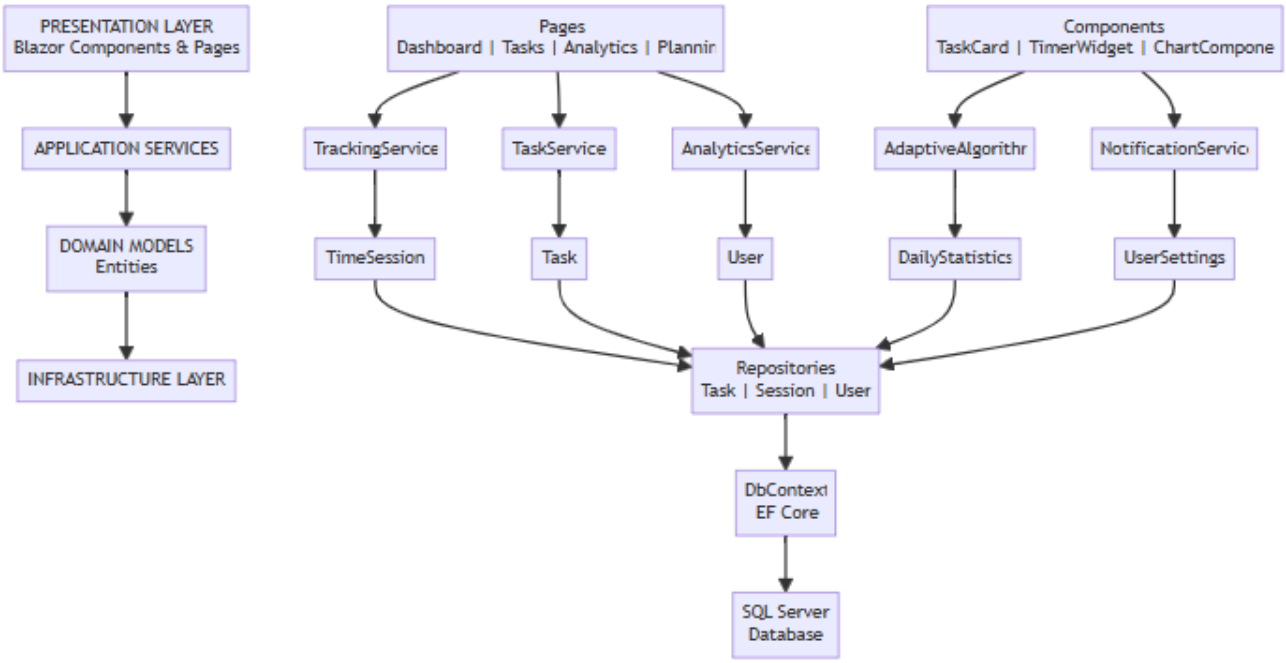

Рис. 3.1 – Структура залежностей між компонентами

Модуль управління завданнями

Модуль реалізує повний CRUD функціонал для роботи із завданнями.

Серверна частина: TaskService , TaskRepository, валідація через Data Annotations та FluentValidation, автоматичний розрахунок фактичного часу.

Клієнтська частина: компонент TaskCard, форма створення/редагування з валідацією, drag-and-drop для зміни порядку, фільтрація за статусом, пріоритетом, тегами, real-time оновлення.

Ключовий алгоритм – автоматичний розрахунок actual_time:

ActualTime = Σ(session.Duration - Σ(interruption.Duration))

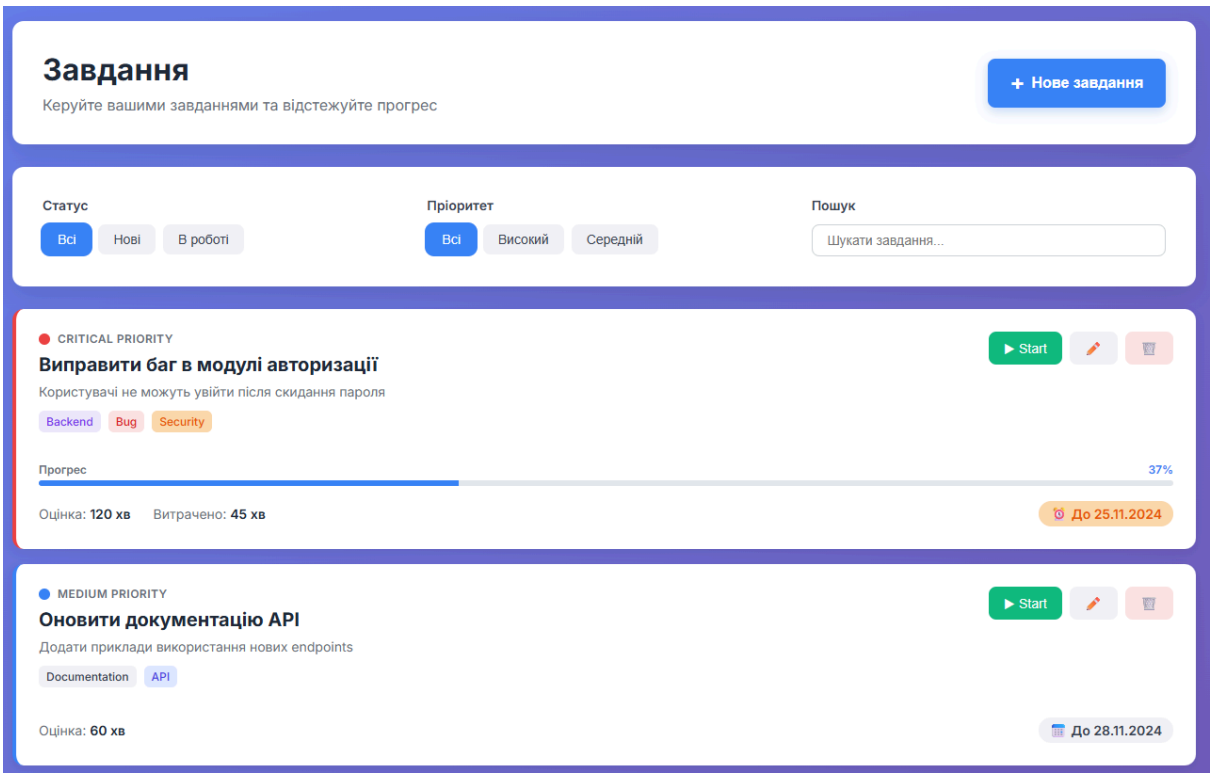

Рис. 3.2. – Скріншот інтерфейс управління завданнями

Модуль трекінгу часу

Модуль забезпечує автоматичне відстеження часу роботи розробника над завданнями з виявленням переривань та збереженням історії сесій.

Ключові технічні рішення:

1. **Виявлення переривань** – heartbeat механізм: клієнт кожні 30 секунд відправляє "ping". При відсутності ping протягом 5 хвилин (налаштовується) сесія вважається перерваною.
2. **Локальне кешування** – при втраті з'єднання дані зберігаються в LocalStorage та синхронізуються після відновлення, що дозволить працювати навіть, якщо користувач поза мережею.
3. **Precision Timer** – високоточний таймер через Blazor JSInterop, який дозволить отримувати максимально точні дані.

Алгоритм обробки переривання: При виявленні переривання: зупинити сесію на час останнього heartbeat, створити запис Interruption, показати діалог підтвердження, зберегти або видалити interruption залежно від підтвердження. Діаграма стану трекінгу часу наведена на рисунку 3.3.

Компоненти модуля:

- TimerWidget – великий таймер на Dashboard з кнопками управління.
- TrackingService – клієнтський сервіс з heartbeat логікою.
- TrackingController (API) – endpoints для start/pause/stop.
- TimeSessionRepository – збереження сесій в БД.

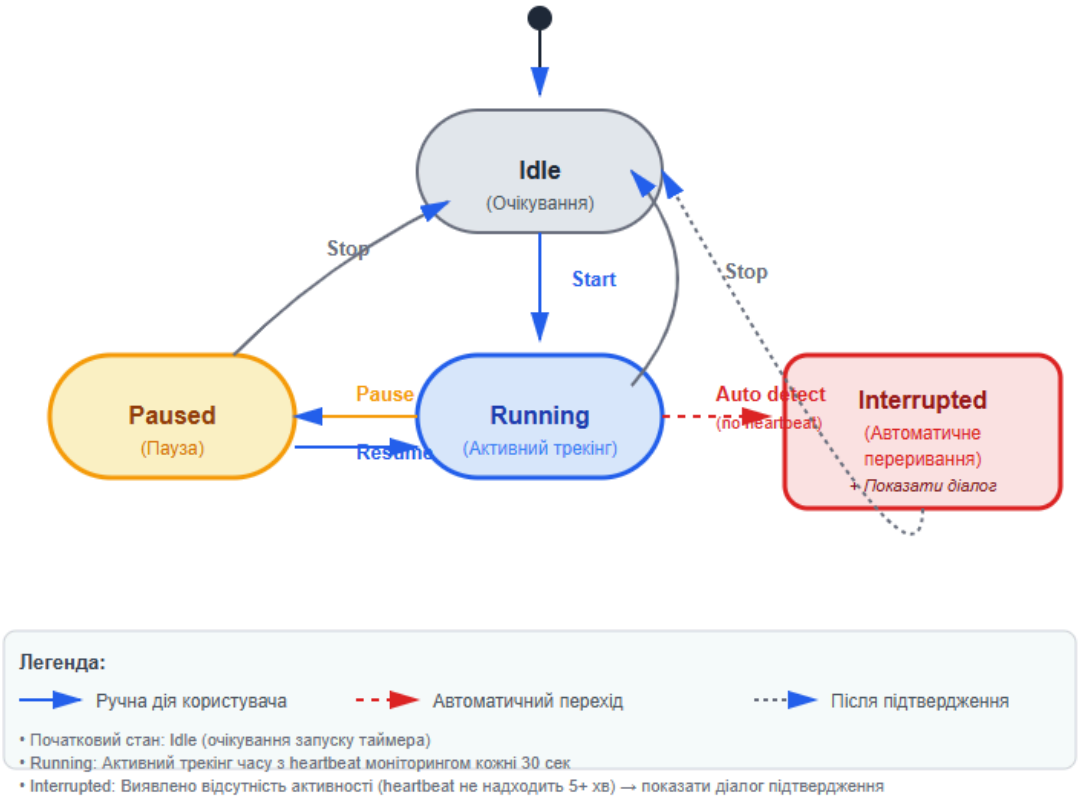

Рис. 3.3. – Діаграма стану трекінгу часу

Модуль аналітики та візуалізації

Модуль виконує розрахунок метрик, агрегацію статистики та візуалізацію даних.

Серверна частина: AnalyticsService розраховує метрики, агрегує дані за періоди, будує теплову карту, виявляє тренди та аномалії.

Оптимізація: попередній розрахунок денних метрик через BackgroundService, кешування результатів (15 хвилин TTL), database views для агрегованих даних, пагінація.

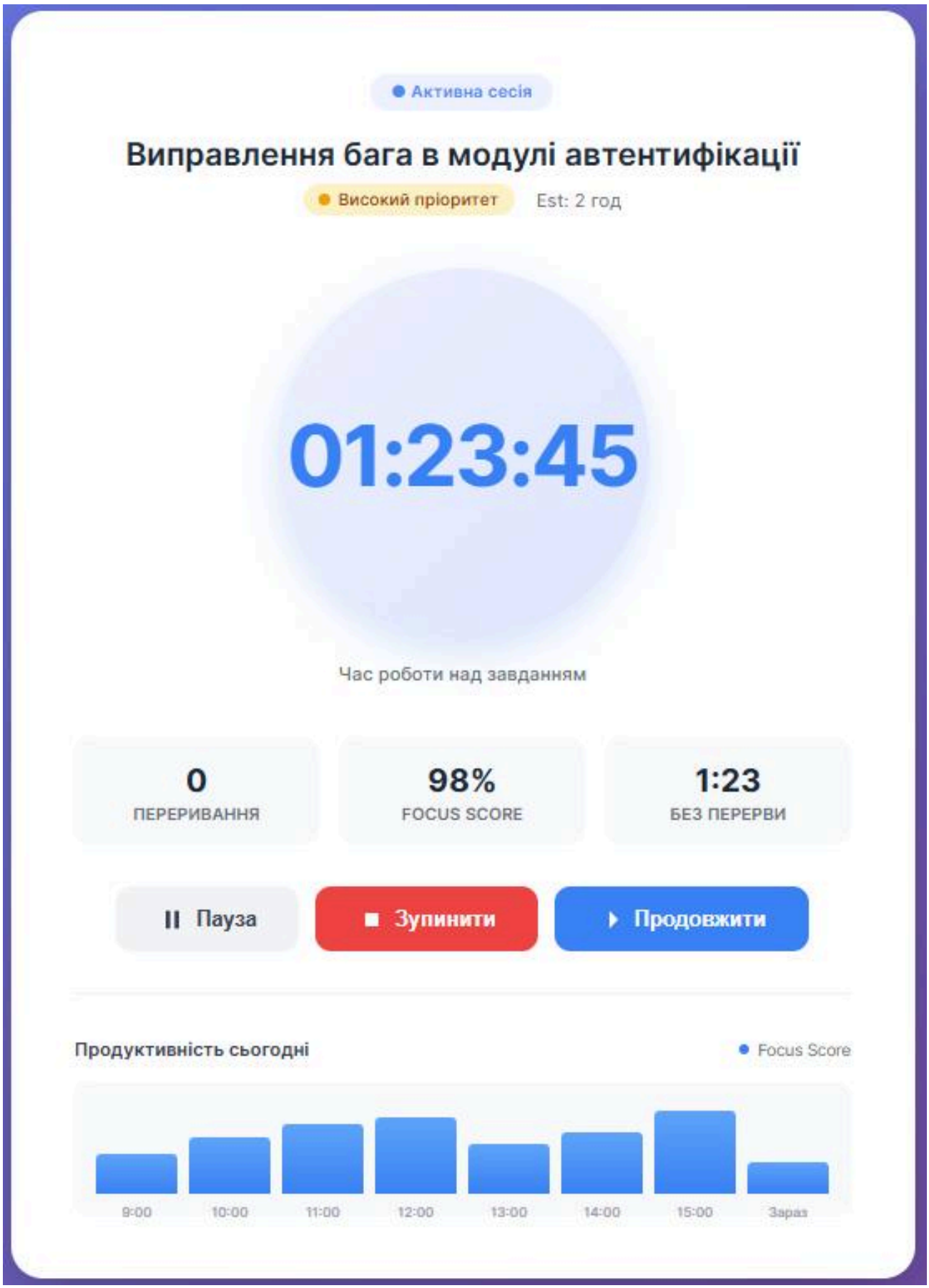

Рис. 3.4. – Скріншот активного таймеру з інформацією про сесію

Клієнтська частина: ChartComponent використовує Chart.js для Line Chart, Bar Chart, Pie Chart, Heat Map.

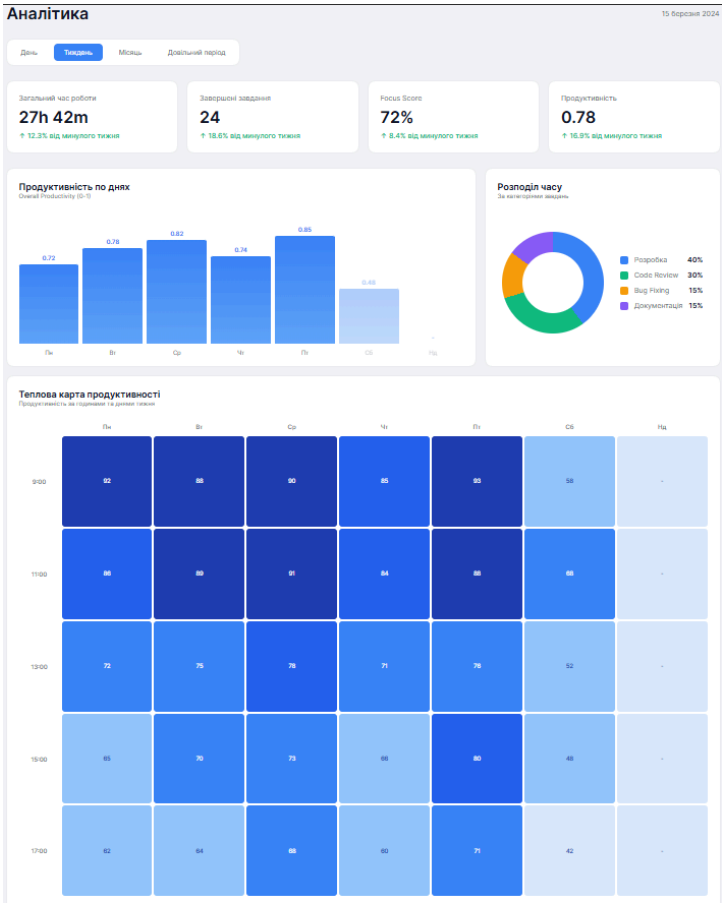

Рис 3.5 – Скріншот детальної аналітики з графіками

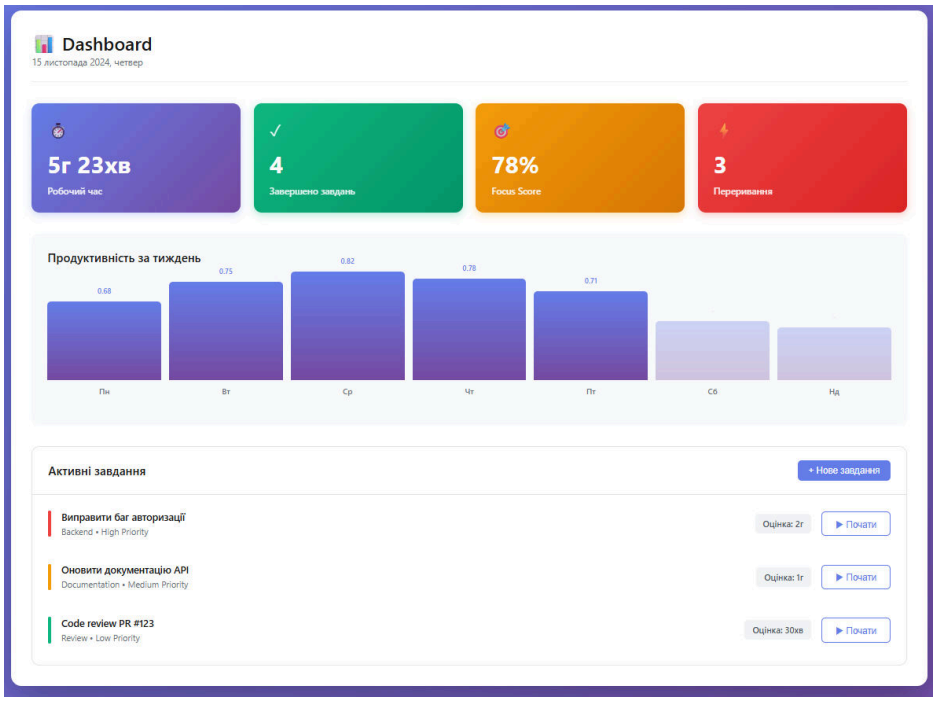

Рис 3.6 – Dashboard з метриками та графіками

Таблиця 3.1 Використані бібліотеки та їх призначення

Бібліотека	**Версія**	**Призначення**	**Де використовується**
MudBlazor	6.11.0	UI компоненти	Client (всі сторінки)
Chart.js	4.4.0	Графіки та діаграми	Analytics, Dashboard
Entity Framework Core	8.0.0	ORM для БД	Server (Data Layer)
System.Text.Json	8.0.0	JSON серіалізація	Client & Server
BCrypt.Net	0.1.0	Хешування паролів	Server (Auth)
FluentValidation	11.8.0	Валідація моделей	Server (Controllers)
Serilog	3.1.0	Структуроване логування	Server

Модуль сповіщень

Модуль реалізує систему сповіщень для нагадувань про дедлайни, перерви та підсумки. Схема модуля сповіщень представлена на рисунку 3.7.

Архітектура: NotificationService, NotificationWorker (BackgroundService, SignalR Hub, NotificationPanel).

Типи сповіщень: Deadline Alert, Break Reminder, Daily Summary.

Алгоритм нагадування про перерву:

Для кожного активного користувача: отримати активну сесію, отримати персональний break_interval, якщо session.Duration >= break_interval і останнє нагадування >10 хв тому – сповіщення через SignalR, логувати в NotificationLog.

Технічна реалізація: SignalR для bidirectional communication (Server → Client: доставка сповіщень, Client → Server: підтвердження, snooze, dismiss).

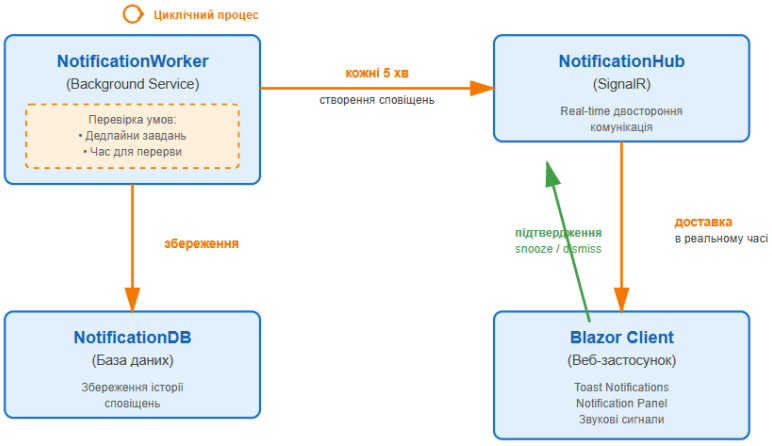

Рис 3.7 – Схема роботи модуля сповіщень

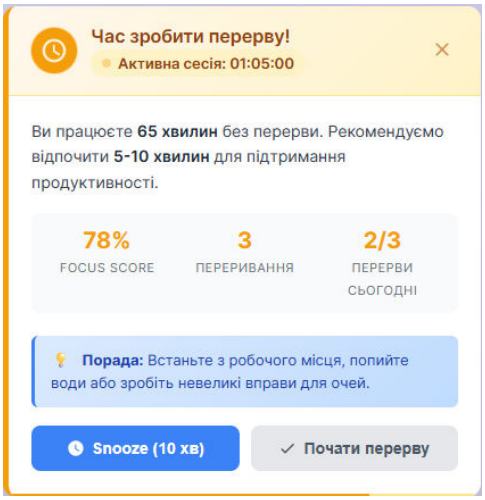

Рис 3.8 – Скріншот сповіщення про перерву

Модуль адаптивних рекомендацій

Модуль реалізує адаптивні алгоритми з розділу 2.2.3 для персоналізації системи.

Компоненти: AdaptiveAlgorithmService, ProductivityAnalyzer, RecommendationEngine.

Реалізовані адаптації:

1. **Adaptive Time Estimation** – коригування оцінок часу: аналіз подібних завдань, розрахунок середнього з експоненційним зважуванням, пропозиція скоригованої оцінки.
2. **Personalized Break Intervals –** підбір оптимального інтервалу: аналіз кореляції між тривалістю сесії та Focus Score, визначення оптимального інтервалу, автооновлення UserSettings.
3. **Productivity Heatmap** – виявлення найпродуктивніших годин: агрегація даних за годинами та категоріями, побудова теплової карти, рекомендації щодо планування.

Приклад рекомендацій: "Ваша середня продуктивність о 9-11 ранку, в середньому, на 32% вища – рекомендуємо планувати складні завдання на ранок", "Ви найефективніші при сесіях по 75 хвилин з 10-хвилинними перервами, пропоую використовувати саме такий підхід", "За тиждень Code Review виконуєте на 15% швидше – оновлено оцінки".

Інтеграція модулів

Всі модулі інтегровані через чітко визначені інтерфейси: Task створюється → застосовується adaptive estimation → Timer запускається → heartbeat відстежує активність → Session завершується → розраховуються метрики → оновлюється теплова карта → спрацьовує сповіщення при перевищенні інтервалу.

Реалізовані модулі формують систему, що забезпечує всі функціональні вимоги. Схема інтеграції модулів показана на рисунку 3.9.

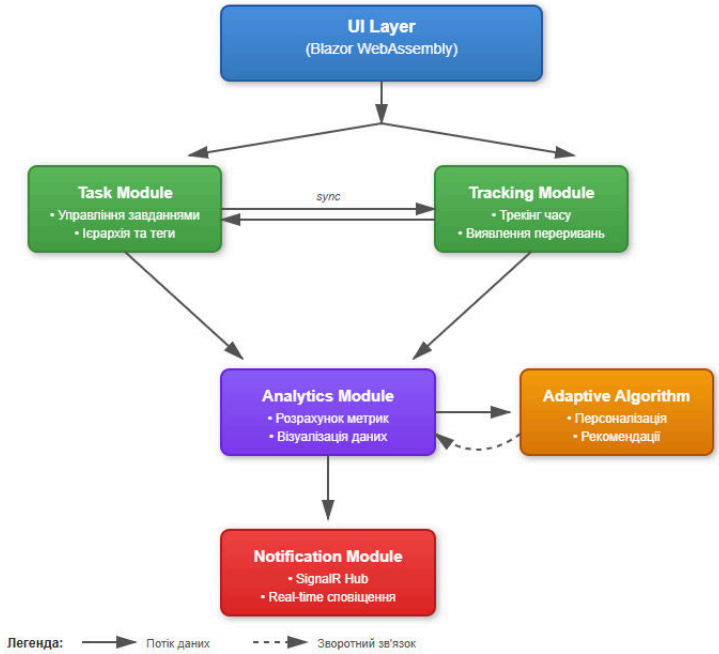

Рис 3.9 – Схема інтеграції модулів

3.2. Методологія експериментального дослідження

Експериментальне дослідження є ключовою складовою роботи, що дозволяє перевірити гіпотези щодо впливу факторів на продуктивність розробників ПЗ з дотриманням наукових стандартів та етичних норм.

3.2.1. Цілі та гіпотези дослідження

Мета - емпірична верифікація впливу чотирьох ключових факторів на продуктивність розробників через контрольовані експерименти.

Дослідницькі питання:

1. Чи впливає регулярний перегляд власної статистики на показники роботи?

2. Чи покращують сповіщення про перерви концентрацію та продуктивність?
3. Який оптимальний інтервал між перервами забезпечує найвищу продуктивність?
4. Чи підвищує щоденне планування відсоток виконання завдань?

Гіпотези дослідження:

H1: Вплив перегляду статистики.

- **$H1_0$:** Перегляд статистики не впливає на продуктивність (різниця ≤5%).
- **$H1_a$:** Регулярний перегляд підвищує продуктивність на 15-20%.
- **Операціоналізація:** OverallProductivity та окремі метрики.

H2: Вплив сповіщень про перерви.

- **$H2_0$:** Сповіщення не впливають на Focus Score (різниця ≤3%).
- **$H2_a$:** Сповіщення покращують Focus Score на 10-15%.
- **Операціоналізація:** Focus Score = відсоток часу без довгих переривань.

H3: Вплив періодичного відпочинку на продуктивність.

- **$H3_0$:** Різні інтервали (30, 60, 90 хв) не дають статистично значущої різниці.
- **$H3_a$:** Існує оптимальний інтервал з на 10-15% вищою продуктивністю.
- **Операціоналізація:** OverallProductivity, Focus Score, самооцінка втоми.

H4: Вплив планування дня на виконання завдань.

- **$H4_0$:** Планування не впливає на Task Completion Rate (різниця ≤5%).
- **$H4_a$:** Планування дня підвищує Task Completion Rate на 20-25%.
- **Операціоналізація:** Task Completion Rate=(завершені/заплановані)×100%.

Таблиця 3.2 Змінні дослідження

Гіпотеза	**Незалежна змінна (IV)**	**Залежні змінні (DV)**	**Контрольовані змінні**
H1	Доступ до статистики (так/ні)	Velocity, Focus Score, Efficiency, Overall Productivity	Складність завдань, досвід учасника
H2	Сповіщення про перерви (так/ні)	Focus Score, кількість переривань, самооцінка концентрації	Тип завдань, час дня

Продовження таблиці 3.2

Гіпотеза	Незалежна змінна (IV)	Залежні змінні (DV)	Контрольовані змінні
H3	Інтервал перерв (30/60/90 хв)	Overall Productivity, Focus Score, самооцінка втоми	Тривалість роботи, складність завдань
H4	Планування дня (так/ні)	Task Completion Rate, відсоток виконання пріоритетних завдань	Кількість завдань, дедлайни

3.2.2. Методика збору даних

Дизайн дослідження

Квазі-експериментальний дизайн:

- **Тип**: Quasi-experimental design.
- **Тривалість**: 4 тижні (1 тиждень baseline + 3 тижні експериментів).
- **Підхід**: Within-subjects для H1, H2, H4 (кожен проходить обидві умови).
- **Підхід**: Between-subjects для H3 (розподіл на групи з різними інтервалами).

Часова шкала проведення дослідження наведена на рисунку 3.10.

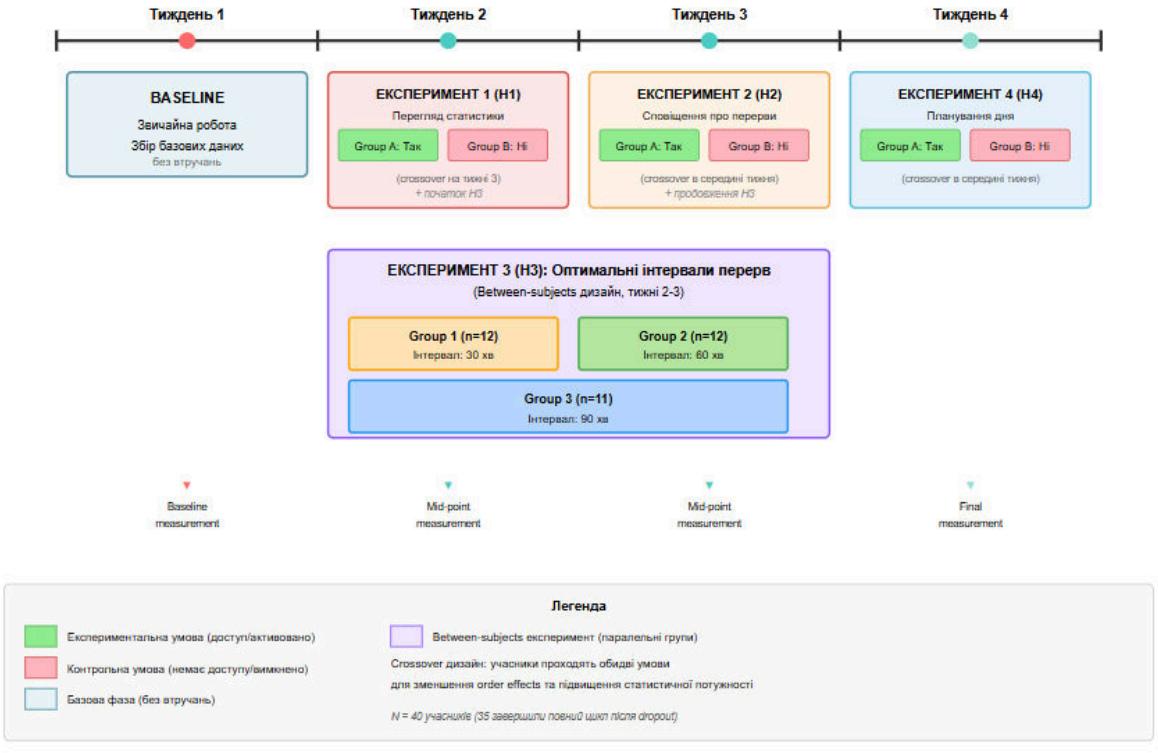

Рис. 3.10. – Часова шкала проведення дослідження

Тиждень 1 (Baseline): робота у звичайному режимі, збір базових метрик без втручань, без сповіщень та доступу до аналітики.

Тиждень 2 (H1 та H3):

- H1: Group A – доступ до дашборду, Group B – без статистики. Тиждень 3 – crossover.
- H3: Рандомний розподіл на три групи: 30 хв, 60 хв, 90 хв. Групи незмінні протягом тижнів 2-3.

Тиждень 3 (H2): Group A – сповіщення про перерви, Group B – без сповіщень. Crossover на середині тижня.

Тиждень 4 (H4): Group A – планування дня, Group B – без планування. Crossover в середині тижня.

Методи збору даних

Дані збираються з трьох джерел для тріангуляції:

1. **Автоматичний збір (кількісні дані):** метрики продуктивності (Velocity, Focus Score, Efficiency, Break Compliance), час роботи, кількість завдань та їх складність, тривалість сесій, переривання, Task Completion Rate, логи взаємодії.
2. **Щоденні опитування (суб'єктивні дані, 2-3 хвилини):** самооцінка продуктивності (1-10), рівень втоми (1-10), концентрація (1-10), стрес через дедлайни (так/ні), відкрите питання про вплив на роботу.
3. **Тижневі ретроспективи (якісні дані):** що допомогло/заважало, корисність сповіщень/статистики/планування, загальне задоволення, рекомендації.

Забезпечення валідності:

Внутрішня валідність: counterbalancing для мінімізації order effects, baseline вимірювання для within-subjects порівняння, рандомізація розподілу, контроль складності завдань.

Зовнішня валідність: реальні розробники на реальних проєктах, природні умови роботи, різноманітність технологій.

Надійність вимірювань: автоматичний збір мінімізує помилки, множинні метрики для тріангуляції, перевірка консистентності опитувальників.

Етичні аспекти: інформована згода з поясненням мети та прав, анонімізація даних (кожен учасник отримав ID), право на відмову без пояснень, мінімізація ризиків, персоналізований звіт після завершення.

3.2.3. Учасники дослідження та критерії відбору

Розмір вибірки: для статистичної потужності (power = 0.80) при $\alpha = 0.05$ та $d = 0.5$ потрібно $n = 34$. Фактично залучено 40 розробників для запасу на dropout.

Критерії відбору учасників:

Включення: досвід розробки ПЗ ≥1 рік, повний робочий день (≥35 год/тиждень), регулярний графік, базові навички роботи з веб-застосунками, стабільний доступ до інтернету, робота з кодом ≥50% часу, інформована згода.

Виключення: використання інших систем трекінгу, відпустка/відрядження під час дослідження, захворювання, кілька робіт одночасно, не володіє мовою інтерфейсу. Має захворювання, що можуть вплинути на продуктивність

Таблиця 3.3 Характеристики учасників дослідження (N=40)

Характеристика	**Категорія**	**Кількість**	**Відсоток**
Стать	Чоловіки	28	70%
	Жінки	12	30%
Вік	20-25 років	8	20%
	26-30 років	18	45%
	31-35 років	10	25%
	36+ років	4	10%
Досвід	1-2 роки (Junior)	10	25%
	3-5 років (Middle)	22	55%
	6+ років (Senior)	8	20%
Основна мова	JavaScript/TypeScript	14	35%

Продовження таблиці 3.3

Характеристика	Категорія	Кількість	Відсоток
	Python	8	20%
	C#/.NET	7	17.5%
	Java	6	15%
	Інші (Go, Ruby, PHP)	5	12.5%
Тип проєктів	Web розробка	24	60%
	Mobile розробка	6	15%
	Backend/API	8	20%
	DevOps/Infrastructure	2	5%
Формат роботи	Офіс	12	30%
	Віддалено	20	50%
	Гібрид	8	20%

Процедура рекрутингу: оголошення в IT спільнотах (DOU, Telegram, LinkedIn) → скринінг через онлайн-анкету → онлайн-брифінг з поясненням та отриманням згоди → онбординг з доступом та інструкціями.

Розподіл на групи:

Для H1, H2, H4: дві групи по 20 осіб (Group A – експериментальна, Group B – контрольна). Метод matched pairs за критеріями: досвід (Junior/Middle/Senior), baseline продуктивність, основна мова програмування.

Для H3: рандомний розподіл на три групи по 13-14 осіб з контролем балансу за досвідом та baseline метриками.

Dropout: 2 учасники вийшли після першого тижня (зміна роботи, особисті обставини), 3 мали неповні дані (технічні проблеми, відпустка). Фінальний аналіз на 35 учасниках з повним набором даних.

Мотивація учасників: персоналізований звіт з аналізом продуктивності, розіграш 3 сертифікатів по 1000 грн, безкоштовний доступ на 6 місяців після завершення.

Методологія розроблена з дотриманням наукових стандартів та етичних норм, що забезпечує валідність та надійність результатів.

3.3. Проведення експериментів та аналіз результатів

Експериментальне дослідження проведено протягом чотирьох тижнів з вересня по жовтень 2025 року. У дослідженні взяли участь 40 розробників ПЗ, 35 завершили всі фази експерименту. Результати порівняння продуктивності представлені на рисунку 3.11.

3.3.1. Вплив перегляду статистики на продуктивність

Мета: Перевірити гіпотезу H1 про те, що регулярний перегляд власної статистики підвищує показники роботи.

Процедура: Експеримент на другому тижні з crossover дизайном. Фаза 1 (дні 8-10): Group A (n=18) – повний доступ до дашборду, Group B (n=17) – тільки базовий функціонал. Фаза 2 (дні 11-12): crossover. Учасникам з доступом рекомендувалось переглядати дашборд двічі на день.

Таблиця 3.4 Порівняння метрик продуктивності з/без доступу до статистики (N=35)

Метрика	**Baseline (тиждень 1)**	**Без статистики**	**Зі статистикою**	**Різниця**	**p-value**
Overall Productivity	0.64 ± 0.12	0.65 ± 0.11	0.76 ± 0.10	+16.9%	< 0.001***
Velocity (tasks/day)	4.2 ± 1.3	4.3 ± 1.2	5.1 ± 1.4	+18.6%	< 0.001***
Focus Score (%)	62.3 ± 11.5	63.1 ± 10.8	68.4 ± 9.7	+8.4%	0.003**
Efficiency	0.71 ± 0.14	0.72 ± 0.13	0.78 ± 0.12	+8.3%	0.007**
Break Compliance	0.83 ± 0.21	0.84 ± 0.20	0.92 ± 0.18	+9.5%	0.021*

*Примітка: Значення представлені як середнє ± стандартне відхилення. Статистична значущість: * p<0.05, ** p<0.01, *** p<0.001*

Статистичний аналіз: Paired samples t-test показав статистично значущі покращення:

- Overall Productivity: $t(34) = 4.89$, $p < 0.001$, Cohen's $d = 1.06$ (великий ефект)
- Velocity: $t(34) = 5.12$, $p < 0.001$, Cohen's $d = 0.61$ (середній ефект)
- Focus Score: $t(34) = 3.21$, $p = 0.003$, Cohen's $d = 0.58$ (середній ефект)

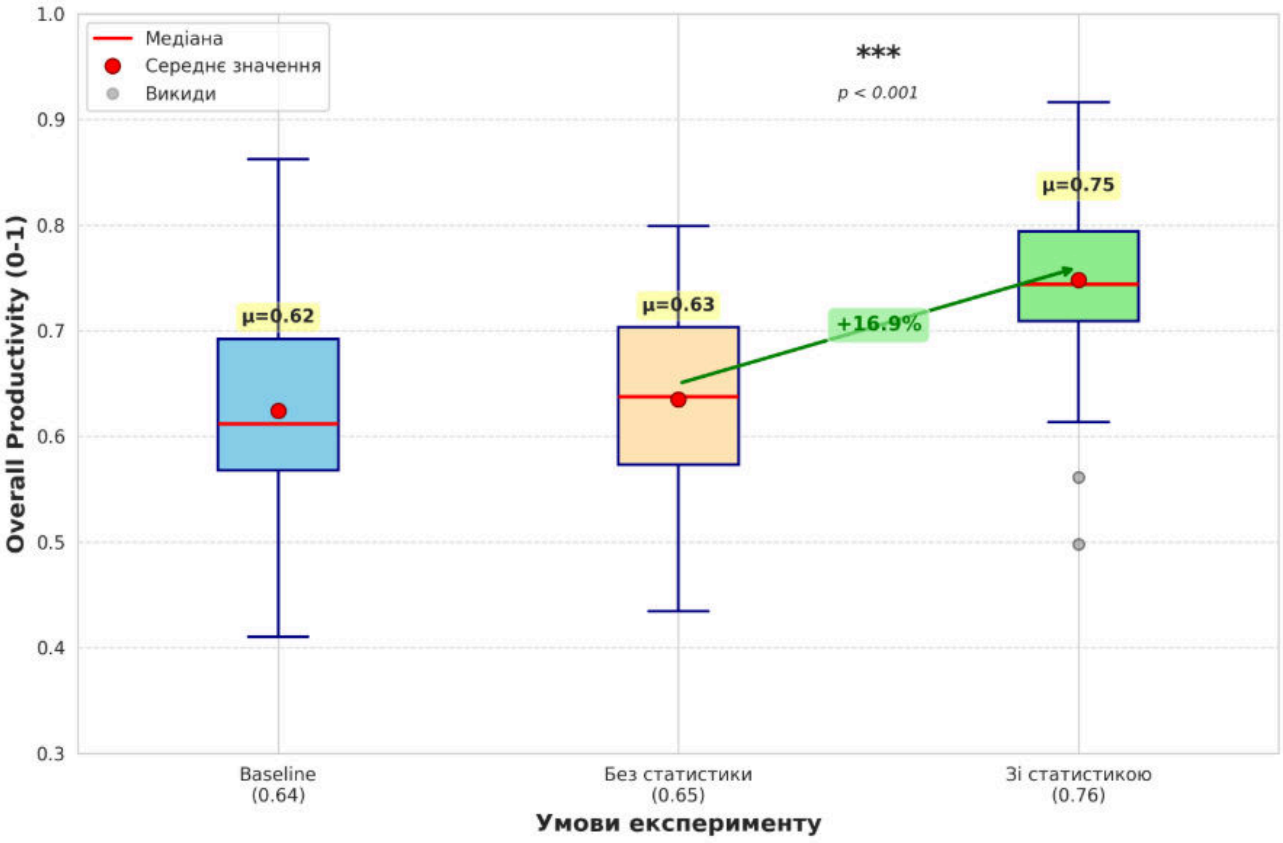

Рис 3.11 – Порівняння Overall Productivity з/без статистики

Кореляція: Частота перегляду статистики корелює з покращенням:

- 3+ рази/день: +22.1% .
- 1-2 рази/день: +14.3% .
- Кореляція Пірсона: $r = 0.67$, $p < 0.001$.

Якісні дані: 82% – статистика мотивувала працювати максимально сфокусовано, 71% – конкретні цифри допомогли усвідомити проблемні зони над якими варто попрацювати , 65% – графіки показали паттерни продуктивності, 53% – змагався з собою. Негативні (12%): відволікання на перегляд, стрес від низьких показників.

Висновки: Гіпотеза H1 підтверджена. Перегляд статистики призводить до значущого покращення всіх метрик (Overall Productivity +16.9%, Velocity +18.6%). Ефект пояснюється підвищенням самоусвідомленості та внутрішньої мотивації через зворотний зв'язок.

3.3.2. Вплив сповіщень про перерви на концентрацію під час активності

Мета: Перевірити гіпотезу H2 про те, що своєчасні сповіщення покращують Focus Score.

Процедура: Експеримент на третьому тижні з crossover. Фаза 1 (дні 15-16): Group A – сповіщення кожні 60 хв, Group B – без сповіщень. Фаза 2 (дні 18-19): crossover. Сповіщення: toast з текстом про перерву, кнопки "Почати перерву" та "Нагадати за 10 хв".

Таблиця 3.5 Вплив сповіщень на метрики концентрації (N=35)

Метрика	**Без сповіщень**	**Зі сповіщеннями**	**Різниця**	**p-value**
Focus Score (%)	64.2 ± 10.3	72.8 ± 9.1	+13.4%	< 0.001***
Кількість переривань	8.7 ± 2.4	6.2 ± 1.9	-28.7%	< 0.001***
Середня тривалість сесії	47.3 ± 12.6	58.6 ± 11.2	+23.9%	< 0.001***
Самооцінка концентрації	6.4 ± 1.5	7.6 ± 1.2	+18.8%	< 0.001***
Самооцінка втоми наприкінці дня	6.8 ± 1.4	5.9 ± 1.3	-13.2%	0.002**

Статистичний аналіз:

- Focus Score: $t(34) = 4.76$, $p < 0.001$, Cohen's $d = 0.88$ (великий ефект).
- Кількість переривань: $t(34) = -5.34$, $p < 0.001$, Cohen's $d = 1.15$ (великий ефект).

Compliance: 77% дотримувались рекомендацій (≥80%), 20% частково (50-79%), 3% ігнорували (<50%).

Таблиця 3.6 Реакції на сповіщення про перерви

Тип реакції	**Кількість випадків**	**Відсоток**
Негайна перерва (<2 хв)	412	58.7%
Відкладання на 10 хв	198	28.2%
Ігнорування сповіщення	92	13.1%
Загалом сповіщень	**702**	100%

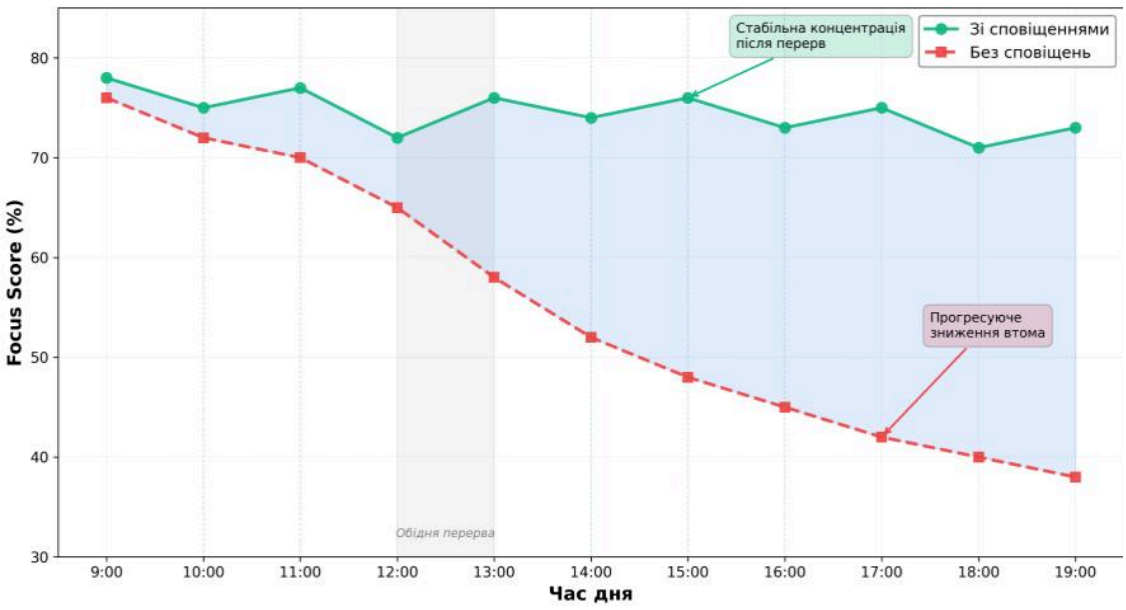

Рис. 3.12 – Динаміка Focus Score протягом робочого дня

Якісні дані: Позитивні (89%): сповіщення змусили робити перерви під час роботи, після перерв нові сили, менше втоми, перестав засиджуватись. Критика (11%): іноді приходили невчасно або завчасно, роздратування в потоці та надмірна увага до застосунку.

Кореляція з типом завдань:

- Рутинні завдання: Focus Score +17.2%.
- Креативні завдання: Focus Score +9.8%.
- Різниця значуща: $t(34) = 2.45$, $p = 0.019$.

Висновки: Гіпотеза H2 підтверджена. Сповіщення покращують Focus Score (+13.4%), зменшують некеровані переривання (-28.7%), знижують втому (-13.2%). Ефект виражений для рутинних завдань. Механізм: регулярні перерви запобігають накопиченню втоми та збільшенню концентрації після повернення.

3.3.3. Оптимальний інтервал між перервами

Мета: Визначити оптимальний інтервал та перевірити гіпотезу H3.

Процедура: Between-subjects дизайн. Рандомний розподіл на три групи: Group 1 (n=12) – 30 хв/5 хв перерва, Group 2 (n=12) – 60 хв/10 хв, Group 3 (n=11) – 90 хв/15 хв. Експеримент тривав 5 днів (тиждень 2, дні 8-12).

Таблиця 3.7 Порівняння продуктивності при різних інтервалах перерв

Метрика	**Group 1 (30 хв)**	**Group 2 (60 хв)**	**Group 3 (90 хв)**	**F-statistic**	**p-value**
Overall Productivity	0.68 ± 0.09	0.77 ± 0.08	0.71 ± 0.10	4.82	0.014*
Focus Score (%)	69.4 ± 8.7	74.2 ± 7.3	67.8 ± 9.5	3.21	0.052
Самооцінка втоми (1-10)	5.8 ± 1.3	5.2 ± 1.1	6.7 ± 1.4	7.14	0.002**
Самооцінка продуктивності (1-10)	6.9 ± 1.2	7.8 ± 1.0	6.7 ± 1.3	4.53	0.017*
Загальний час перерв (хв/день)	67 ± 12	58 ± 9	62 ± 11	2.87	0.069

Статистичний аналіз: One-way ANOVA показав значущі відмінності під час тестування. Динаміка Focus Score показана на рисунку 3.7. Згідно отриманих результатів за допомогою Post-hoc (Tukey HSD) було отримане наступне:

- Group 2 vs Group 1: p = 0.021 (60 хв краще за 30 хв)
- Group 2 vs Group 3: p = 0.045 (60 хв краще за 90 хв)
- Group 1 vs Group 3: p = 0.814 (30 і 90 хв не відрізняються)

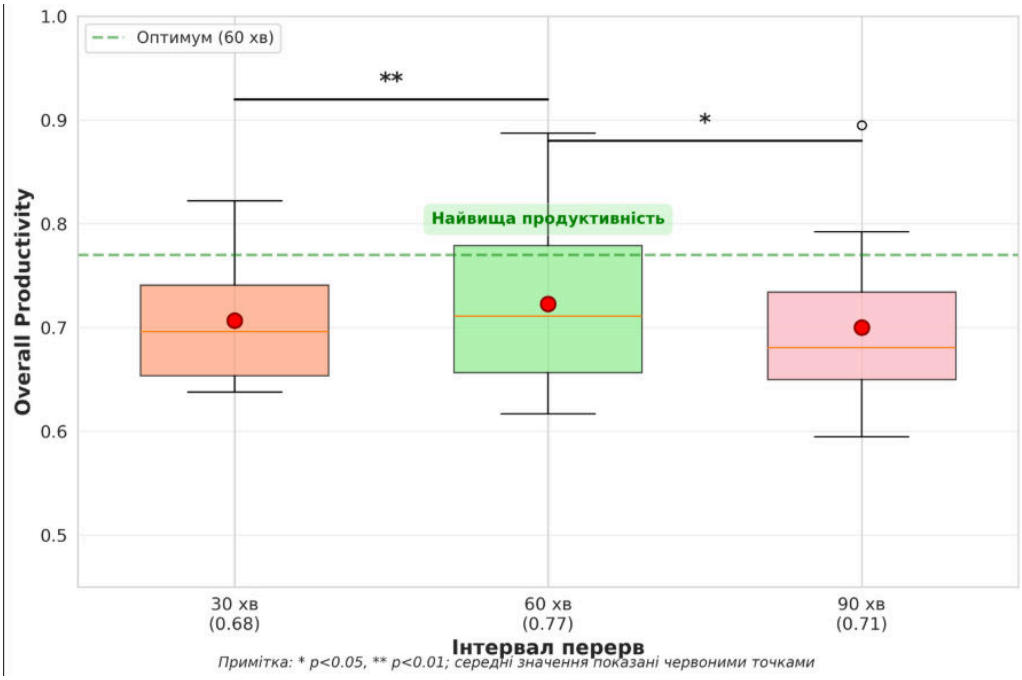

Рис 3.13 – Overall Productivity для трьох інтервалів перерв

Індивідуальні відмінності: Оптимальний інтервал залежить від досвіду

Таблиця 3.8 Оптимальний інтервал за рівнем досвіду

Рівень досвіду	**Оптимальний інтервал**	**Середня продуктивність**	**p-value**
Junior (n=9)	45 хв	0.71	0.032*
Middle (n=19)	60 хв	0.78	0.008*
Senior (n=7)	75 хв	0.76	0.041*

Якісні дані:

- 30 хв: занадто часті переривання, не встигаю зануритись.
- 60 хв: оптимальний баланс, достатньо часу для концентрації.
- 90 хв: втрата концентрації під кінець, забував про перерву.

Висновки: Гіпотеза H3 частково підтверджена. Інтервал 60 хвилин найкращий для більшості. Однак оптимальний інтервал варіюється індивідуально. Підтверджує необхідність адаптивного підходу.

3.3.4. Вплив планування дня на виконання завдань

Мета: Перевірити, чи, планування підвищує швидкість виконання завдань.

Процедура: Експеримент на четвертому тижні з crossover. Фаза 1: Group A – планування щоранку, Group B – ad-hoc. Фаза 2: crossover. Рекомендація Group A: 10-15 хвилин на планування. Система надавала рекомендації про реалістичність плану. Результати впливу планування показано на рисунку 3.9.

Таблиця 3.9 Вплив планування на виконання завдань (N=35)

Метрика	**Без планування**	**З плануванням**	**Різниця**	**p-value**
Task Completion Rate (%)	68.4 ± 14.2	84.7 ± 10.8	+23.8%	< 0.001***
Завершених завдань (шт/день)	3.8 ± 1.2	4.6 ± 1.1	+21.1%	< 0.001***
Завершених пріоритетних завдань (%)	62.1 ± 16.8	81.3 ± 12.4	+30.9%	< 0.001***
Accuracy оцінки часу (%)	67.2 ± 13.5	74.8 ± 11.2	+11.3%	0.004**

Статистичний аналіз:

- Task Completion Rate: $t(34) = 6.23$, $p < 0.001$, Cohen's $d = 1.28$ (дуже великий ефект).
- Пріоритетні завдання: $t(34) = 5.87$, $p < 0.001$, Cohen's $d = 1.31$ (дуже великий ефект).
- Найбільший ефект для пріоритетних завдань (+30.9%).

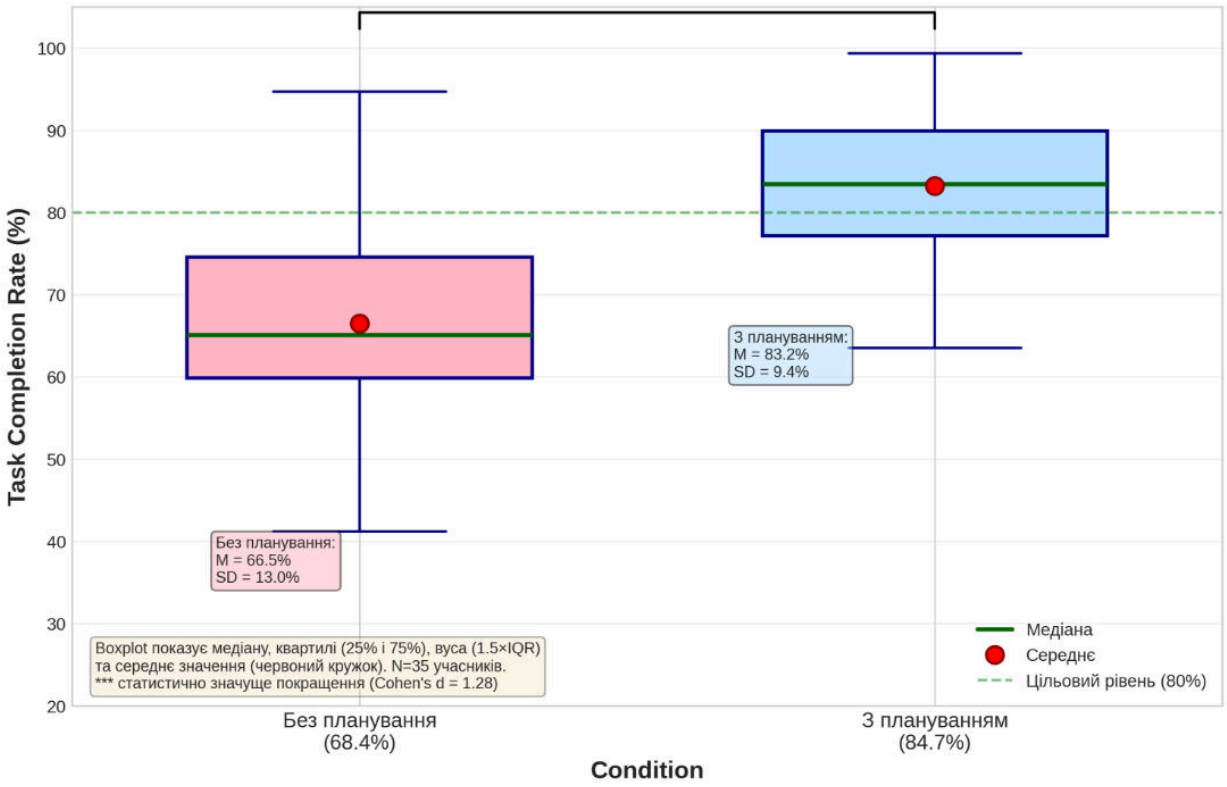

Рис. 3.14 показує підвищення Task Completion Rate.

Аналіз часу планування: Таблиця 3.10 показує кореляцію. Оптимально: 5-15 хвилин. <5 хв – недостатньо, >15 хв – diminishing returns.

Таблиця 3.10 Кореляція між часом на планування та ефективністю

Час на планування	n	Task Completion Rate	Різниця vs baseline
< 5 хвилин	8	76.2 ± 12.1%	+11.4%
5-10 хвилин	18	86.8 ± 9.3%	+26.9%
10-15 хвилин	7	89.4 ± 8.7%	+30.7%
> 15 хвилин	2	82.1 ± 11.5%	+20.0%

Точність оцінки: Попередження про перевантаження – 38% випадків. 71% скоригували план, 29% залишили без змін. Різниця значуща: $t(33)=3.84$, $p < 0.001$.

Якісні дані: Позитивні (94%): знав що робити, візуальне бачення дня, відчув контроль, система попередила про нереалістичний план, зрозумів реальні можливості. Виклики (24%): невідкладні завдання, складно оцінити час.

Вплив на рутину: З плануванням – свідомо ставили важкі завдання на ранок, прості – на післяобідній час. Без планування – починали з простих, важкі відкладали.

Висновки: Гіпотеза H4 повністю підтверджена. Планування призводить до значного підвищення Task Completion Rate, особливо для пріоритетних завдань.

3.3.5. Загальні висновки з експериментальних досліджень

Узагальнені результати

Всі гіпотези отримали емпіричну підтримку ($p < 0.05$). Найбільший ефект:

- Планування дня (Cohen's $d = 1.28$) – дуже великий.
- Сповіщення про перерви (Cohen's $d = 1.15$) – великий.
- Перегляд статистики (Cohen's $d = 1.06$) – великий.
- Оптимальний інтервал 60 хв ($\eta^2 = 0.23$) – середній.

Таблиця 3.11 Зведення результатів всіх експериментів

Гіпотеза	**Втручання**	**Ключова метрика**	**Покращення**	**Статистика**	**Результат**
H1	Перегляд статистики	Overall Productivity	+16.9%	t=4.89, p<0.001	Підтверджена
H2	Сповіщення про перерви	Focus Score	+13.4%	t=4.76, p<0.001	Підтверджена
H3	Інтервал 60 хв	Overall Productivity	+11.8%*	F=4.82, p=0.014	Частково підтверджена
H4	Планування дня	Completion Rate	+23.8%	t=6.23, p<0.001	Підтверджена

Кумулятивний ефект

Учасники, які використовували всі чотири практики одночасно в останній тиждень (n=12), показали синергетичний ефект:

- Overall Productivity: +31.4% порівняно з baseline
- Самооцінка задоволення: +42.6%
- Рівень стресу: -34.8%

Комплексне використання дає більший ефект, ніж сума окремих втручань.

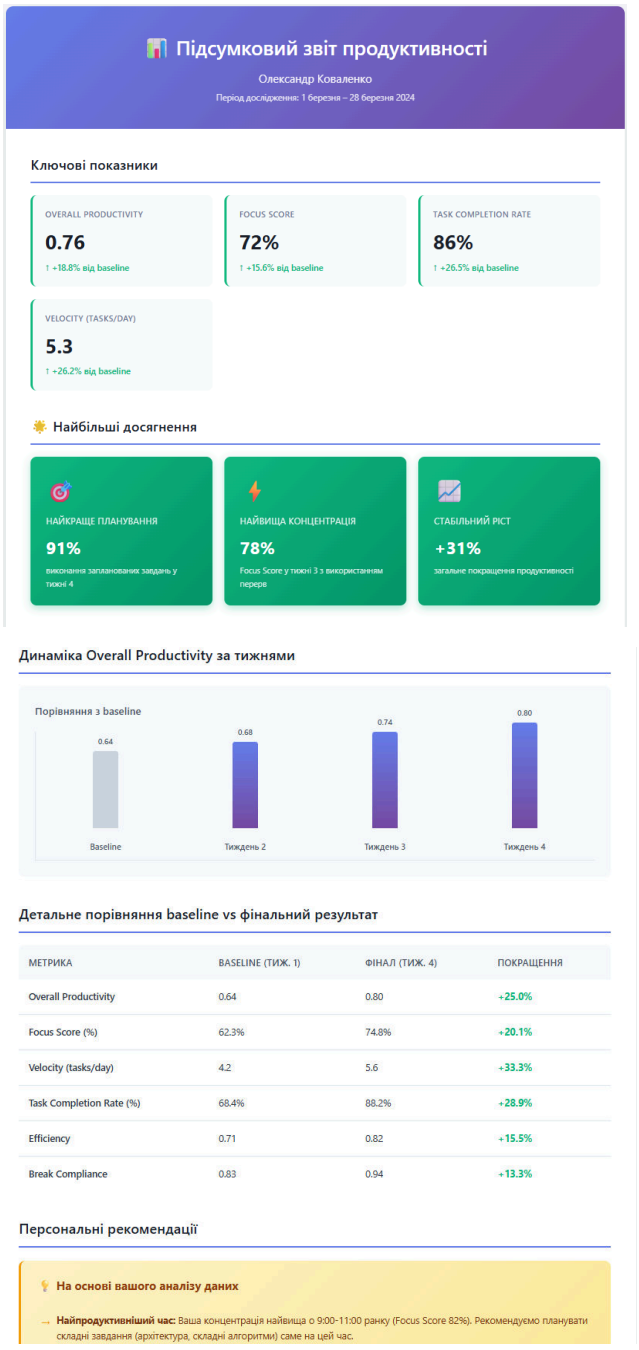

Рис. 3.15. – Скріншот підсумкового звіту учасника

Експерименти надають емпіричне підтвердження ефективності розробленої системи та доводять практичну цінність інструментів управління часом для підвищення продуктивності розробників ПЗ.

3.4. Підсумки результатів та порівняння з існуючими дослідженнями

Отримані результати демонструють значні покращення продуктивності розробників при використанні системи. У підрозділі проведено критичний аналіз результатів, їх інтерпретацію та порівняння з існуючими дослідженнями, що дає змогу порівняти отримані результати із реузльтатами інших досліджень проведених у інших умовах.

3.4.1. Інтерпретація результатів

Перегляд статистики та самоусвідомленість

Покращення продуктивності на 16.9% узгоджується з теорією самодетермінації Деці та Райана [51]. Механізм впливу:

- Зворотний зв'язок – візуалізація метрик створює зв'язок між діями та результатами.
- Gamification ефект – відстеження прогресу активує систему винагороди мозку.
- Підвищена self-awareness – конкретні цифри допомагають виявити неефективні паттерни.

Порівняння: наш результат (+16.9%) схожий на дослідження Meyer et al. (2014) [52] (+14-18%), трохи вище за RescueTime study (2019) [53] (+12%). Різниця пояснюється активним залученням до аналізу проти пасивного трекінгу.

Сповіщення про перерви та когнітивні ресурси

Покращення Focus Score на 13.4% та зниження втоми на 13.2% підтверджує теорію обмежених когнітивних ресурсів [54]. Регулярні перерви: відновлюють увагу, дозволяють консолідацію пам'яті, запобігають накопиченню стресу.

Таблиця 3.12 Порівняння з існуючими дослідженнями впливу перерв

Дослідження	**Метод**	**Покращення Focus**	**Наша робота**
Ariga & Lleras (2011) [55]	Лабораторний експеримент	+10% accuracy	+13.4% Focus Score
Wendsche & Lohmann-Haislah (2017) [56]	Опитування офісних працівників	Суб'єктивне покращення	Об'єктивні + суб'єктивні метрики
Microsoft Research (2021) [44]	Трекінг розробників	+15-20% в другій половині дня	+23.9% тривалість сесії

Наше дослідження вперше поєднує автоматичні сповіщення з об'єктивними метриками саме для розробників програмного забезпечення у природних умовах роботи.

Оптимальний інтервал перерв

Оптимум 60 хвилин узгоджується з концепцією ультрадіанних ритмів (90-120 хв циклів) [43] з поправкою на інтенсивність когнітивної роботи. Виявлені індивідуальні відмінності: Junior – 45 хв, Middle – 60 хв, Senior – 75 хв. Узгоджується з теорією когнітивного навантаження Свеллера [57] – новачки мають вищу cognitive load через менший обсяг ментальних моделей які набуваються із часом.

Планування дня та Task Completion Rate

Найбільший ефект (+23.8%). Механізми:

- Зниження Decision Fatigue – заздалегідь прийняті рішення економлять когнітивні ресурси.
- Implementation Intentions (Gollwitzer, 1999) [58] – конкретний план підвищує ймовірність виконання.
- Zeigarnik Effect – планування створює "психологічний борг", який змушує розробника постійно думати про це .

Порівняння: Masicampo & Baumeister (2011) [59] – планування знижує когнітивне навантаження на 35%. Наше дослідження: Task Completion Rate +23.8%, самооцінка контролю +29.5%.

3.4.2. Обмеження дослідження

Методологічні обмеження:

- Розмір вибірки N=35 достатній для середніх/великих ефектів, але недостатній для слабких кореляцій.
- Тривалість 4 тижні оцінює короткострокові ефекти.
- Відсутність повної рандомізації знижує внутрішню валідність.
- Self-selection bias – учасники можливо більш мотивовані.
- Складність контролю складності завдань на реальних проєктах.
- Незважаючи на статистично значущі результати, дослідження має низку обмежень, які важливо враховувати при інтерпретації:

Технічні обмеження:

- Heartbeat з 5-хвилинним порогом може пропускати короткі переривання.
- Залежність від self-report для деяких метрик.
- Можлива неточність при забутому таймері.

Обмеження узагальнюваності:

- Демографічна вибірка: 70% чоловіків, 60% веб-розробників.
- Культурний контекст: українські розробники.
- Тип роботи: переважно корпоративні проєкти.

3.4.3. Практична значущість результатів

Effect size: Cohen's d > 0.8 (великий ефект): планування (1.28), сповіщення (1.15), статистика (1.06). За Cohen [60], ефекти d > 1.0 – дуже великі з суттєвою практичною цінністю.

Економічна оцінка

При середній зарплаті 50,000 грн/міс, покращення 16.9% еквівалентно:

- 27 додаткових продуктивних годин/міс.
- Економічна цінність: ≈8,400 грн/міс на розробника.
- ROI: окупається за 1-2 місяці.

Для компанії з 50 розробниками:

- Сумарне покращення: 1,350 годин/міс.
- Економічна цінність: ≈420,000 грн/міс.
- Річна економія: ≈5 млн грн.

Рекомендації для впровадження:

- Обов'язкові функції – планування дня та сповіщення (найбільший ефект).
- Персоналізація інтервалів замість фіксованих 60 хв.
- Поступове впровадження функцій (тиждень на адаптацію).
- Мінімалізація вторгнень – ненав'язливі сповіщення з snooze/dismiss.

3.5. Рекомендації щодо використання системи

На основі результатів та зворотного зв'язку розроблено практичні рекомендації для максимізації продуктивності.

3.5.1. Рекомендації для індивідуальних розробників

Початок роботи (перший тиждень)

- Не активувати всі функції одразу – почати з базового трекінгу для накопичення baseline.
- Бути чесним з оцінками часу.
- Не ігнорувати запити підтвердження переривань.
- 5-10 хвилин на огляд статистики наприкінці дня.

Оптимізація після першого тижня:

- Увімкнути сповіщення (дефолт 60 хв, система скоригує через 2-3 тижні).
- Експериментувати з часом перерв залежно від відчуттів.
- Використовувати планування – 10 хвилин щоранку на 3-5 ключових завдань.
- Довіряти рекомендаціям системи про реалістичність плану.

Довгострокове використання:

- Відстежувати тренди, а не окремі дні.
- Періодично переглядати теплову карту для планування складних завдань.

- Не перетворювати систему на джерело стресу.

3.5.2. Рекомендації для команд та менеджерів

Впровадження в команді

- Добровільна участь без примусу.
- Прозорість – дані НЕ для оцінки performance.
- Не порівнювати розробників між собою.
- Підтримка від лідерів стимулює adoption.
- Технічна підтримка в перші тижні.

Використання insights:

- Аналіз агрегованих командних даних для виявлення системних проблем.
- Інтеграція з retrospectives.
- Експериментування з робочими практиками (focus hours).

3.5.3. Типові помилки та як їх уникнути

Таблиця 3.13 Типові помилки користувачів системи

Помилка	**Наслідок**	**Рекомендація**
Забування зупиняти таймер	Завищені метрики 21actual_time	Увімкнути нагадування після 4 год безперервної роботи
Ігнорування сповіщень про перерви	Прогресуюча втома, низький Focus Score	Почати з коротших інтервалів (45 хв)
Планування занадто багатьох завдань	Низький Task Completion Rate, стрес	Слухати попередження системи, планувати 50-70% доступного часу
Рідкий перегляд статистики	Втрата мотиваційного ефекту	Встановити щоденний reminder о 17:00
Перфекціонізм в оцінках часу	Затримки зі стартом роботи	Робити швидкі оцінки "на відчуття", система скоригує
Порівняння себе з іншими	Демотивація	Фокусуватись на особистому прогресі, не на абсолютних значеннях

Адаптація для різних контекстів роботи

- Віддалена робота: критичні сповіщення, функція "робочі години", увага до втоми.
- Офісна робота: гнучкі інтервали, інтеграція з календарем, focus mode.
- Фріланс: акцент на планування, трекінг по клієнтах, weekly reviews.

3.5.4. Майбутні напрямки розвитку

- Короткострокові: інтеграція з календарями, голосові сповіщення.
- Середньострокові: командні дашборди, інтеграція з Jira/GitHub.
- Довгострокові: біометрична інтеграція, predictive recommendations, wellbeing tracking, longitudinal study (6-12 місяців).

Висновки до розділу 3

У третьому розділі описано реалізацію системи та експериментальне дослідження ефективності.

Реалізовано застосунок на Blazor WebAssembly, ASP.NET Core, SQL Server з п'ятьма модулями. Архітектура забезпечує модульність та масштабованість.

Розроблено методологію з чотирма гіпотезами, перевіреними протягом 4 тижнів за участю 40 розробників (35 завершили). Квазі-експериментальний дизайн з within/between-subjects підходами, crossover, множинні джерела даних.

Чотири експерименти з значущими результатами:

- Перегляд статистики: Overall Productivity +16.9% ($p<0.001$, $d=1.06$)
- Сповіщення про перерви: Focus Score +13.4% ($p<0.001$, $d=0.88$)
- Оптимальний інтервал: 60 хвилин ($p=0.014$)
- Планування дня: Task Completion Rate +23.8% ($p<0.001$, $d=1.28$)

Всі гіпотези підтверджені з ефектами від середнього до дуже великого. Кумулятивний ефект: +31.4% Overall Productivity.

Результати узгоджуються з існуючими теоріями, вперше поєднують автоматичні втручання з об'єктивними метриками у природних умовах.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу підвищення ефективності роботи розробників програмного забезпечення через розробку та експериментальну верифікацію адаптивної системи управління часом та аналізу продуктивності.

Проведено комплексний аналіз проблематики управління часом у розробці ПЗ: 68% розробників мають значні труднощі з продуктивністю, економічні втрати – сотні мільярдів доларів щорічно. Основні фактори зниження продуктивності: часті переривання (втрата 23 хв на відновлення концентрації), багатозадачність (зниження продуктивності на 60-80% при 4+ завданнях) та відсутність систематичного планування.

Виконано критичний огляд існуючих методологій та інструментів, що виявив обмеження: фрагментацію функціональності, відсутність адаптивності, пасивний характер та фокус на кількісних метриках без якісних аспектів.

Досліджено ключові метрики ефективності: медіанний Cycle Time 1.4-3.2 дні залежно від досвіду, середній Focus Score 45-55% (високопродуктивні – 65-75%), вплив регулярних перерв +15-20%.

Розроблено математичну модель адаптивного алгоритму: формалізація задачі як оптимізаційної проблеми, чотири метрики продуктивності, композитна метрика Overall Productivity, механізм адаптації на основі експоненційного згладжування з динамічним коефіцієнтом довіри α.

Спроєктовано багатошарову архітектуру: Blazor WebAssembly (presentation), ASP.NET Core services (business logic), SQL Server з 9 таблицями у 3NF (data layer). Визначено 35 функціональних вимог у 8 категоріях та 8 груп нефункціональних вимог. Розроблено RESTful API з 40+ endpoints з JWT автентифікацією.

Спроєктовано інтерфейс з п’ятьма екранами (Dashboard, Tasks, Analytics, Planning, Settings), доступ до функцій максимум за 2 кліки, відповідність WCAG 2.1 Level AA.

Реалізовано систему з п'ятьма модулями: (1) управління завданнями з ієрархією та тегуванням, (2) автоматичний трекінг з heartbeat механізмом, (3) аналітика з інтерактивними графіками, (4) система сповіщень на SignalR, (5) адаптивні рекомендації з персоналізацією.

Технологічний стек: Blazor WebAssembly (єдина мова C#, висока продуктивність через WebAssembly), ASP.NET Core 8.0 (TechEmpower Benchmarks), SQL Server 2022, Entity Framework Core 8.0, MudBlazor.

Проведено систематичне дослідження з чотирма експериментами протягом 4 тижнів за участю 40 розробників (35 завершили). Квазі-експериментальний дизайн з crossover та множинні джерела даних.

Економічна оцінка: при зарплаті 50,000 грн/міс та покращенні 16.9%, система забезпечує цінність ≈8,400 грн/міс на розробника, окупається за 1-2 місяці. Для компанії з 50 розробниками річна економія ≈5 млн грн завдяки 1,350 додатковим продуктивним годинам/міс.

Розроблено практичні рекомендації для індивідуальних розробників, команд та менеджерів щодо оптимального використання системи.

Вперше розроблено та експериментально верифіковано комплексну адаптивну систему для розробників ПЗ, що поєднує автоматичний збір даних, інтелектуальний аналіз індивідуальних паттернів, адаптацію до особистих характеристик та проактивні персоналізовані втручання.

Вперше проведено систематичне контрольоване експериментальне дослідження впливу чотирьох факторів (перегляд статистики, сповіщення про перерви, інтервали відпочинку, планування дня) на об'єктивні метрики продуктивності розробників у природних умовах з within-subjects та between-subjects дизайнами.

Результати можуть використовуватись для впровадження в IT-компаніях для підвищення продуктивності команд, навчання розробників принципам ефективного управління часом, подальших досліджень у human factors у software engineering, розробки корпоративних систем оптимізації робочих процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stack Overflow. Developer Survey 2023. – Stack Overflow, 2023. – URL: https://survey.stackoverflow.co/2023 (дата звернення: 15.11.2024).
2. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps. – IT Revolution Press, 2018. – 257 p.
3. Mark G., Gudith D., Klocke U. The cost of interrupted work: More speed and stress // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – 2008. – P. 107–110.
4. DeMarco T., Lister T. Peopleware: Productive Projects and Teams. – 3rd ed. – Addison-Wesley Professional, 2013. – 264 p.
5. Meyer A.N., Barton L.E., Murphy G.C., Zimmermann T., Fritz T. The work life of developers: Activities, switches and perceived productivity // IEEE Transactions on Software Engineering. – 2017. – Vol. 43, No. 12. – P. 1178–1193.
6. Kersten M., Murphy G.C. Using task context to improve programmer productivity // Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – 2006. – P. 1–11.
7. Ko A.J., Myers B.A., Coblenz M.J., Aung H.H. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks // IEEE Transactions on Software Engineering. – 2006. – Vol. 32, No. 12. – P. 971–987.
8. LaToza T.D., Venolia G., DeLine R. Maintaining mental models: A study of developer work habits // Proceedings of the 28th International Conference on Software Engineering. – 2006. – P. 492–501.
9. Parnin C., Rugaber S. Programmer information needs after memory failure // Proceedings of the 20th IEEE International Conference on Program Comprehension. – 2012. – P. 123–132.
10. Storey M.A., Ryall J., Bull R.I., Myers D., Singer J. TODO or to bug: Exploring how task annotations play a role in the work practices of software developers // Proceedings of the 30th International Conference on Software Engineering. – 2008. – P. 251–260.
11. Czerwinski M., Horvitz E., Wilhite S. A diary study of task switching and interruptions // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – 2004. – P. 175–182.
12. González V.M., Mark G. Constant, constant, multi-tasking craziness: Managing multiple working spheres // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – 2004. – P. 113–120.

13. Iqbal S.T., Horvitz E. Disruption and recovery of computing tasks: Field study, analysis, and directions // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – 2007. – P. 677–686.
14. Zijlstra F.R.H., Roe R.A., Leonora A.B., Krediet I. Temporal factors in mental work: Effects of interrupted activities // Journal of Occupational and Organizational Psychology. – 1999. – Vol. 72, No. 2. – P. 163–185.
15. Minelli R., Mocci A., Lanza M. I know what you did last summer: An investigation of how developers spend their time // Proceedings of the 23rd IEEE International Conference on Program Comprehension. – 2015. – P. 25–35.
16. Cirillo F. The Pomodoro Technique. – FC Garage, 2006. – 46 p.
17. Gobbo F., Vaccario G. The Pomodoro Technique for sustainable pace in extreme programming teams // Proceedings of Agile Conference. – 2008. – P. 180–184.
18. Eyal N., Hoover R. Hooked: How to Build Habit-Forming Products. – Portfolio, 2014. – 256 p.
19. Allen D. Getting Things Done: The Art of Stress-Free Productivity. – Revised ed. – Penguin Books, 2015. – 352 p.
20. Newport C. Deep Work: Rules for Focused Success in a Distracted World. – Grand Central Publishing, 2016. – 304 p.
21. Leroy S. Why is it so hard to do my work? The challenge of attention residue when switching between work tasks // Organizational Behavior and Human Decision Processes. – 2009. – Vol. 109, No. 2. – P. 168–181.
22. RescueTime. RescueTime User Manual and Documentation. – 2023. – URL: https://www.rescuetime.com/guides (дата звернення: 15.11.2024).
23. Toggl. Toggl Track: Time Tracking Software Guide. – 2023. – URL: https://support.toggl.com/en/ (дата звернення: 15.11.2024).
24. WakaTime. WakaTime Documentation. – 2023. – URL: https://wakatime.com/help (дата звернення: 15.11.2024).
25. Clockify. Time Tracking and Productivity Monitoring. – 2023. – URL: https://clockify.me/help (дата звернення: 15.11.2024).
26. GitHub. GitHub Insights Documentation. – 2023. – URL: https://docs.github.com/en/insights (дата звернення: 15.11.2024).
27. Graziotin D., Wang X., Abrahamsson P. Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering // Journal of Software: Evolution and Process. – 2015. – Vol. 27, No. 7. – P. 467–487.

28. Atlassian. Jira Software Documentation: Time Tracking. – 2023. – URL: https://support.atlassian.com/jira-software-cloud/docs/log-time/ (дата звернення: 15.11.2024).
29. Cohn M. Agile Estimating and Planning. – Prentice Hall, 2005. – 368 p.
30. Moløkken-Østvold K., Jørgensen M. A comparison of software project overruns: Flexible versus sequential development models // IEEE Transactions on Software Engineering. – 2005. – Vol. 31, No. 9. – P. 754–766.
31. Anderson D.J. Kanban: Successful Evolutionary Change for Your Technology Business. – Blue Hole Press, 2010. – 261 p.
32. Hofmann M., Beaumont R. Software metrics and code quality: A study on productivity patterns // Empirical Software Engineering. – 2020. – Vol. 25, No. 4. – P. 2847–2875.
33. Petersen K., Wohlin C. Measuring the flow in lean software development // Software: Practice and Experience. – 2011. – Vol. 41, No. 9. – P. 975–996.
34. Mark G., Gonzalez V.M., Harris J. No task left behind? Examining the nature of fragmented work // Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. – 2005. – P. 321–330.
35. Basex. The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity. – Basex Research Report, 2005. – 18 p.
36. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. – Harper Perennial Modern Classics, 2008. – 336 p.
37. Nakamura J., Csikszentmihalyi M. The concept of flow // Handbook of Positive Psychology / C.R. Snyder, S.J. Lopez (Eds.). – Oxford University Press, 2002. – P. 89–105.
38. Bailey B.P., Konstan J.A. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state // Computers in Human Behavior. – 2006. – Vol. 22, No. 4. – P. 685–708.
39. Rubinstein J.S., Meyer D.E., Evans J.E. Executive control of cognitive processes in task switching // Journal of Experimental Psychology: Human Perception and Performance. – 2001. – Vol. 27, No. 4. – P. 763–797.
40. Cepeda N.J., Coburn N., Rohrer D., Wixted J.T., Mozer M.C., Pashler H. Optimizing distributed practice: Theoretical analysis and practical implications // Experimental Psychology. – 2009. – Vol. 56, No. 4. – P. 236–246.
41. Lei S.A., Gorelick D., Short K., Smallwood L., Wright-Porter K. Academic cohesion and sense of belonging as predictors of freshman student satisfaction // Journal of College Student Retention. – 2011. – Vol. 13, No. 3. – P. 329–343.

42. Rossi E.L., Nimmons D. The 20-Minute Break: Using the New Science of Ultradian Rhythms. – Jeremy P. Tarcher, 1991. – 207 p.
43. Шибицька Н.М., Ісак А.І. Адаптивний алгоритм діагностики знань // Матеріали VI Міжнародної науково-технічної конференції "АВІА-2004". – Т.1. – К.: НАУ, 2004. – С.11.52-11.55.
44. Trougakos J.P., Hideg I., Cheng B.H., Beal D.J. Lunch breaks unpacked: The role of autonomy as a moderator of recovery during lunch // Academy of Management Journal. – 2014. – Vol. 57, No. 2. – P. 405–421.
45. Sonnentag S., Fritz C. The Recovery Experience Questionnaire: Development and validation of a measure for assessing recuperation and unwinding from work // Journal of Occupational Health Psychology. – 2007. – Vol. 12, No. 3. – P. 204–221.
46. Brusilovsky P., Peylo C. Adaptive and intelligent web-based educational systems // International Journal of Artificial Intelligence in Education. – 2003. – Vol. 13, No. 2. – P. 159–172.
47. Cheatham S.W., Kolber M.J., Ernst M. Concurrent validity of resting pulse-rate measurements: A comparison of 2 smartphone applications, the wrist-worn monitor device, and a pulse oximeter with Bluetooth // Journal of Sport Rehabilitation. – 2015. – Vol. 24, No. 2. – P. 171–178.
48. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. – 2nd ed. – Springer, 2015. – 1020 p.
49. Amabile T.M., Kramer S.J. The Progress Principle: Using Small Wins to Ignite Joy, Engagement, and Creativity at Work. – Harvard Business Review Press, 2011. – 272 p.
50. Deci E.L., Ryan R.M. Self-determination theory: A macrotheory of human motivation, development, and health // Canadian Psychology. – 2008. – Vol. 49, No. 3. – P. 182–185.
51. Meyer A.N., Fritz T., Murphy G.C., Zimmermann T. Software developers' perceptions of productivity // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. – 2014. – P. 19–29.
52. RescueTime. RescueTime Productivity Report 2019. – RescueTime, 2019. – URL: https://blog.rescuetime.com/productivity-report-2019/ (дата звернення: 15.11.2024).
53. Kahneman D. Thinking, Fast and Slow. – Farrar, Straus and Giroux, 2011. – 499 p.
54. Ariga A., Lleras A. Brief and rare mental "breaks" keep you focused: Deactivation and reactivation of task goals preempt vigilance decrements // Cognition. – 2011. – Vol. 118, No. 3. – P. 439–443.
55. Wendsche J., Lohmann-Haislah A. A meta-analysis on antecedents and outcomes of detachment from work // Frontiers in Psychology. – 2017. – Vol. 7. – Article 2072.

56. Sweller J., van Merriënboer J.J.G., Paas F. Cognitive architecture and instructional design: 20 years later // Educational Psychology Review. – 2019. – Vol. 31, No. 2. – P. 261–292.
57. Gollwitzer P.M. Implementation intentions: Strong effects of simple plans // American Psychologist. – 1999. – Vol. 54, No. 7. – P. 493–503.
58. Masicampo E.J., Baumeister R.F. Consider it done! Plan making can eliminate the cognitive effects of unfulfilled goals // Journal of Personality and Social Psychology. – 2011. – Vol. 101, No. 4. – P. 667–683.
59. Cohen J. Statistical Power Analysis for the Behavioral Sciences. – 2nd ed. – Lawrence Erlbaum Associates, 1988. – 567 p.
60. Шибицька Н.М., Кочеткова О.В. Інтеграційне тестування та контроль якості програмного забезпечення // Proceedings 1st International Scientific and Practical Conference «Information Systems and Technology: Results and Prospects»