МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

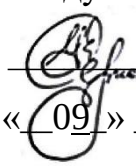
Національний університет кораблебудування імені адмірала Макарова

Навчально науковий інститут комп'ютерних наук та управління проєктами

Кафедра управління проєктами

«Допущений до захисту»

Завідувач кафедри


Чернов С.К.

«_09_» _грудня_ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «**магістр**»

на тему:

Формування та управління командою розробки програмного забезпечення в Agile-проєктах

Виконав: студент 6171М групи

Сокол М.О.

(підпис)

Керівник роботи: д.т.н.,професор


Чернов С. К.

Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

Навчально науковий інститут комп'ютерних наук та управління проєктами
Кафедра управління проєктами
Спеціальність 122 Комп`ютерні науки
Освітня програма «Управління проєктами»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____проф. Чернова Л.С.
(підпис)
« 09 » грудня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТА
на здобуття ступеня вищої освіти **«магістр»**

Студенту ***Соколу Максиму Олександровичу***

1. Тема роботи: Формування та управління командою розробки програмного забезпечення в Agile-проєктах
Керівник роботи д.т.н., професор Чернов С.К. _____
Затверджені наказом ректора № 1406-уч. від «04» грудня 2025 року
2. Термін подання роботи: до 09.12.2025р.
3. Вихідні дані по роботі: фахова література, довідники з управління проєктами, матеріали конференцій, репозитарій університету
4. Перелік питань, що належать до розробки (найменування розділів)
Розділ 1.Теоретичні основи формування та управління командою розробки ПЗ в Agile-проєктах
Розділ 2.Методології управління командами розробки програмного забезпечення
Розділ 3.Формування команди розробки програмного забезпечення в Agile-проєктах
Розділ 4. Охорона праці
Розділ 5. Охорона навколишнього середовища
5. Перелік презентаційних матеріалів виконаний в програмі Power Point
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 4. Охорона праці	Мозговий А.М., доцент		
Розділ 5. Охорона навколишнього середовища	Мозговий А.М., доцент		

7. Дата видачі завдання 16.09.2025р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Вивчення літературних джерел з предмету дослідження	Вересень 2025	виконано
2	Складання розгорнутого плану магістерської роботи та ознайомлення керівника з планом кваліфікаційної роботи	Вересень 2025	виконано
3	Написання розділу 1	Вересень 2025	виконано
4	Написання розділу 2	Жовтень 2025	виконано
5	Написання розділу 3	Листопад 2025	виконано
5	Написання розділу 4	Листопад 2025	виконано
6	Написання розділу 5	Листопад 2025	виконано
7	Оформлення магістерської роботи	Грудень 2025	виконано
8	Передача магістерської роботи рецензенту для рецензування	Грудень 2025	виконано
9	Передача магістерської роботи науковому керівникові для написання відгуку	Грудень 2025	виконано
10	Попередній захист магістерської роботи	09.12.2025	виконано
11	Захист магістерської роботи	25.12.2025	виконано

Студент _____ Сокол М. О.
(підпис)

Керівник роботи _____ Чернов С. К.
(підпис)

АНОТАЦІЯ

Сокол М.О. Формування та управління командою розробки програмного забезпечення в Agile-проєктах. Кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки», освітньої програми «Управління проєктами». Миколаїв: НУК ім. адм. Макарова, 2025. 78 сторінок.

У магістерській роботі розглянуто теоретичні та практичні аспекти формування й управління командою розробки програмного забезпечення в Agile-проєктах. Проаналізовано еволюцію Agile-методологій, їхні цінності та принципи, а також сучасні підходи до організації роботи команд у сфері інформаційних технологій. Практична цінність роботи полягає у можливості використання отриманих результатів і рекомендацій для підвищення ефективності управління командами розробки програмного забезпечення в умовах застосування Agile-методологій.

Ключові слова: Agile, Scrum, команда розробки програмного забезпечення, управління проєктами, самоорганізація, комунікація, IT-команда.

ANNOTATION

Sokol Maksym. Formation and management of the software development team in Agile projects. Qualification work on specialty 122 "Computer Science", educational program "Project Management". Mykolaiv: NUK named after adm. Makarova, 2025. 78 pages.

The master's thesis examines the theoretical and practical aspects of the formation and management of a software development team in Agile projects. The evolution of Agile methodologies, their values and principles, as well as modern approaches to organizing the work of teams in the field of information technologies are analyzed. The practical value of the work lies in the possibility of using the obtained results and recommendations to improve the effectiveness of software development team management in the context of Agile methodologies.

Key words: Agile, Scrum, software development team, project management, self-organization, communication, IT team.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій є надзвичайно динамічним, програмні продукти складні, вимоги користувачів та ринку змінюються. За таких обставин традиційні практики управління проектами, що передбачають фіксоване планування та поступове впровадження етапів, дедалі частіше не працюють ефективно.

Потреба в швидкому реагуванні (на зміни), адаптації (до нових бізнес-цілей) та високій якості настільки критична, що все більше людей звертаються до Agile для гнучких методологій управління. Agile-методи стали важливим інструментом для компаній, які прагнуть підвищити конкурентоспроможність, скоротити час виходу на ринок та створити команди, здатні до самоорганізації та високої продуктивності. Agile базується на цінностях взаємодії, командної роботи, адаптивного планування та швидкої доставки функціонуючого програмного продукту. Тому управління командою в Agile-проектах набуває першочергового значення, оскільки добре управління роботою команди є життєво важливим для успішного результату проекту.

Хоча деякі з особливостей команд в Agile, такі як експерти, є важливими, команда розробників в Agile не є просто групою спеціалістів, а є цілою системою взаємодії, де кожен внесок, ефективна співпраця інших членів команди, своєчасна комунікація, спільні рішення та швидка реакція на змінні обставини є ключовими. Вона враховує професійні можливості членів команди, їх психологічну сумісність та мотиваційні фактори, а також розподіл ролей, що забезпечують ефективне виконання цілей проекту в змінених умовах. Цей проект втілює найбільш актуальні принципи Agile, філософії управління проектами, яка цінує співпрацю, гнучкість та внесок зацікавлених сторін.

Водночас управління Agile-командою є набагато більше, ніж традиційне управління. Короткі спринти використовуються для комунікації Scrum Master, Product Owner та членів команди розробників, підкреслюючи покращення, використання метрик та дух прозорості, відкритості та зворотного зв'язку.

Проте, ці проблеми формування та управління командами залишаються актуальними, незважаючи на зростання методів Agile.

З іншого боку, команди можуть мати прогалини в розподілі ролей, недостатню комунікацію, низьку мотивацію та перевантаження учасників. Це створює ризики зниження продуктивності, затримок у розробці та зниження якості продукту. Через глобалізацію IT-ринку та поширення форматів віддаленої роботи також зростає попит на дослідження механізмів формування Agile-команд.

Однак роль правильно організованих систем комунікації, інструментів координації та методів оцінки продуктивності стає більш важливою, оскільки багато компаній переходять на гібридні або повністю віддалені моделі співпраці. За таких умов вимоги до типу команди, професійних навичок її членів, навичок самопланування та взаємної підтримки стають значно вищими.

Тепер необхідно з'ясувати, як побудувати хорошу Agile-команду, які принципи, інструменти та методології є для покращення Agile-команди, і як організувати процес управління для досягнення стабільного прогресу до передбачуваних результатів. Існує особливе значення в аналізі проблем у реальних проектах та реалізації практичних пропозицій щодо їх усунення.

Оскільки Agile практикується по всьому світу для реалізації проектів у веб-розробці, мобільних додатках, хмарних сервісах, штучному інтелекті, бізнес-аналітиці та інших галузях інформаційних технологій, напрямок досліджень у цій фокусній області є надзвичайно актуальним. Такі практики ефективні у підвищенні продуктивності команди, покращенні якості кінцевих продуктів та управлінні пріоритетами зацікавлених сторін в умовах обмежених ресурсів. Отже, Agile-команди можна визначити як частину основних компетенцій IT-менеджерів проектів, які формують сильні Agile-команди.

Метою магістерської роботи є глибокий аналіз процесу формування та керівництва командою програмної інженерії в Agile-проектах, визначення змінних, що визначають ефективність команди, розробка рекомендацій щодо

покращення взаємодії команди. Для реалізації цієї мети необхідно виконати численні пов'язані завдання для теоретичного аналізу, дослідницьких застосувань та практичної розробки практичного рішення, всі з яких розроблені для вирішення у взаємопов'язаній манері.

Об'єктом дослідження є процес управління командою розробників програмного забезпечення в контексті застосування методологій Agile. Предметом дослідження є методи, підходи, інструменти та організаційні механізми формування та підвищення ефективності Agile-команд. Подальші дослідження включатимуть читання наукових публікацій та джерел робіт, пов'язаних з методологією Agile, управлінням командами, психологією командної роботи та організацією процесу розробки програмного забезпечення.

Наукова новизна роботи полягає в узагальненні та систематизації сучасних підходів до формування й управління командами розробки програмного забезпечення в умовах Agile-проєктів. У роботі уточнено роль командної структури, компетенцій учасників та комунікаційних практик у забезпеченні ефективності Agile-команд, а також визначено взаємозв’язок між організаційною культурою, моделями взаємодії та результативністю командної роботи.

Продуктивність команди Scrum, роль Product Owner як того, хто просуває пріоритизацію, як менеджера команди, роль Scrum Master як менеджера процесу, методи самоорганізації команди та вимірювання продуктивності продукту є темами, які досліджуються в сучасній науковій літературі. Інші вчені також розглядають організаційну культуру, мотивацію, стилі лідерства, відносини та інше, як команда працює зі своїми колегами. Роботи англійською мовою, розроблені провідними професіоналами, включають Кена Швабера, Джеффа Сазерленда, Майка Кона, Романа Піхлера та Крейга Лармана, і служать основою для розуміння як теоретичних основ, на яких має базуватися Agile, так і точних деталей, як команди повинні взаємодіяти.

Ці підходи визначаються українськими дослідниками, які адаптують їх для національного IT-ринку та умов місцевих компаній. Аналіз джерел показує відсутність структури в наукових дослідженнях феномену Agile-командобудування. Існують окремі рекомендації щодо ролей, комунікації та організації процесів, але відсутні комплексні дослідження, що інтегрують теоретичний, практичний та прикладний вимір. Це нагадує про актуальність теми цієї роботи, включаючи необхідність її більш глибокого пояснення.

Для досягнення встановленої мети було виконано:

- Дослідження концептуальної основи понять та принципів Agile-підходів, та принципів побудови команд розвитку як формування.
- Вивчення характеристик управління командами в Agile-командах в Agile-проектах та взаємодії користувачів між зацікавленими сторонами.
- Аналіз в аналітичному дослідженні фактичної або змодельованої Agile-команди, та представлення існуючих проблем та викликів, з якими стикається.
- Agile-команда в фактичному або змодельованому Agile-полі як існуючі проблеми та перешкоди для хорошої продуктивності.
- Надання пропозицій для покращення співпраці, ролей та оптимізації процесів командної роботи (ідеальна продуктивність команди) та ролей, і максимізації функціонування групи.
- Опис можливих моделей управління Agile-командою (модель для кращого управління Agile-командою), які можуть бути інтегровані в систему, та пропозиція можливого методу її впровадження.
- Роздуми про IT-професіоналів у відношенні до безпеки праці та безпеки життя.
- Практичне застосування підходів до формування та управління Agile-командою

Кваліфікаційна робота складається зі вступу, п'ять розділів, висновку, 35 джерел. Перший розділ описує теоретичні підходи до Agile та формування команд. Другий розділ розглядає конкретну команду Agile-проекту. Третій розділ представляє модель, яка сприятиме ефективному управлінню командою. Четвертий та п'ятий розділи присвячені безпеці праці, екологічній безпеці та умовам праці спеціалістів з інформаційних технологій.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ФОРМУВАННЯ ТА УПРАВЛІННЯ КОМАНДОЮ РОЗРОБКИ ПЗ В AGILE-ПРОЄКТАХ

1.1. Сутність та еволюція Agile-методологій

Стрімкий розвиток інформаційних технологій у другій половині XX — на початку XXI століття призвів до появи нових вимог до процесу розробки програмного забезпечення. Класичні моделі, зокрема каскадна (Waterfall), часто виявлялися недостатньо гнучкими, оскільки передбачали жорстку послідовність етапів і обмежені можливості адаптації до змін. У міру зростання складності програмних продуктів та швидкості трансформації бізнес-середовища виникла потреба у нових підходах, здатних забезпечити адаптивність, високу швидкість доставки та залучення команд у процес прийняття рішень.

Перші передумови Agile з’явилися ще у 80–90-х роках XX століття, коли розробники почали відмовлятися від перевантаженої документації та надмірно структурованих моделей на користь гнучких ітеративних підходів. До числа таких попередників належать Spiral Model Баррі Боема, Incremental Model, Rapid Application Development та інші адаптивні концепції. Однак справжнім переломним моментом стало підписання у 2001 році Маніфесту гнучкої розробки програмного забезпечення (Agile Manifesto), який об’єднав основні принципи та цінності нової філософії розробки.

Agile став відповіддю на проблему непередбачуваності проєктів, різких змін вимог користувачів і потребу компаній швидше реагувати на ринкову конкуренцію. На відміну від традиційних підходів, Agile поєднує короткі цикли розробки, орієнтацію на постійну комунікацію та тісну взаємодію між учасниками команди. Його головною метою є створення продукту, який максимально відповідає потребам замовника, а також забезпечення можливості вносити зміни на будь-якому етапі проєкту.

В основі Agile лежить ітеративний підхід: робота над проєктом поділяється на невеликі повторювані цикли, після кожного з яких команда отримує зворотний зв’язок і коригує подальшу діяльність. Це значно зменшує ризики, дозволяє швидко виявляти помилки, адаптувати продукт до мінливих умов та підвищує загальну продуктивність [1].

З часом Agile став не просто набором методологій, а цілою культурою управління проєктами та командної роботи. На його основі виникли популярні фреймворки, такі як Scrum, Kanban, Extreme Programming (XP), Lean Software Development та інші [1]. Кожен з них має власні особливості, однак усі вони дотримуються ключової ідеї: створення цінності для користувача шляхом гнучкої, адаптивної та командоорієнтованої розробки.

Сьогодні Agile широко застосовується не лише в IT, але й у маркетингу, освіті, проєктному менеджменті, стартапах, державному секторі та інших галузях. Це свідчить про універсальність підходу та його здатність підвищувати ефективність роботи команд незалежно від сфери діяльності.

Принциповим аспектом становлення Agile стало усвідомлення того, що традиційні методи управління надмірно зосереджені на процесах, документації та контролі, тоді як успішна розробка програмного продукту значною мірою залежить від людського фактора. Agile-методології запровадили нову парадигму, у якій головну роль відіграють взаємодія між членами команди, спільна відповідальність за результат, швидкі цикли прийняття рішень і здатність до адаптації. Саме люди, їх компетентність, мотивація та здатність до самоорганізації є основою продуктивності команди в межах гнучких підходів [1].

Ще однією важливою рисою еволюції Agile стало поступове відходження від суворої ієрархічної структури управління. У традиційних моделях менеджери часто приймають більшість рішень самостійно, у той час як виконавці виконують поставлені задачі. Agile пропонує протилежний підхід: команда отримує ширші повноваження, активно залучається до планування,

бере участь у визначенні обсягу роботи на кожну ітерацію та самостійно організовує власну діяльність. Такий підхід підвищує рівень відповідальності розробників, покращує якість комунікацій і сприяє більш точному прогнозуванню термінів виконання завдань.

Варто відзначити, що еволюція Agile тісно пов’язана з розвитком ринку стартапів та високотехнологічних компаній. У цих середовищах швидкість виведення продукту на ринок є критично важливою, а невизначеність — природною умовою. Гнучкі методології дозволили цим компаніям ефективно тестувати гіпотези, отримувати зворотний зв’язок від користувачів та адаптувати продукт у режимі реального часу. Це стало одним з ключових факторів, який забезпечив домінування Agile у сучасних підходах розробки.

Крім того, поширення Agile стало можливим завдяки активній спільноті практиків, яка постійно оновлює методики, створює нові фреймворки та надає рекомендації для організації командної роботи. Багато міжнародних організацій, таких як Agile Alliance, Scrum.org, Scrum Alliance, інвестують значні ресурси в дослідження, навчання та сертифікацію фахівців, що сприяє підвищенню професійного рівня команд і менеджерів [1,2,3].

У XXI столітті Agile перетворився на комплексну систему управління, яка охоплює не лише технічні процеси, а й організаційну культуру, комунікації, лідерство та психологію взаємодії. Це робить його універсальним інструментом, що здатний забезпечити високу ефективність команд незалежно від масштабів та специфіки проєктів. Саме тому дослідження сутності та еволюції Agile є фундаментально важливим для подальшого розуміння процесів формування та управління командами розробки програмного забезпечення.

1.2. Цінності та принципи Agile (Agile Manifesto)

Поява Agile-маніфесту у 2001 році стала точкою формалізації підходів гнучкої розробки програмного забезпечення [1]. Документ був створений групою з 17 досвідчених практиків, які прагнули сформувати альтернативу

надмірно бюрократичним та негнучким традиційним методологіям. Чотири ключові цінності та дванадцять принципів визначив Agile Manifesto, які стали ідейним фундаментом усіх гнучких фреймворків, таких як Scrum, Kanban, XP тощо.

Перш за все маніфест підкреслює пріоритет людей та взаємодії над процесами й інструментами. Це означає, що основою успішного виконання проєкту є ефективна комунікація, компетентність членів команди, їхня взаємопідтримка та спроможність працювати як єдиний цілісний механізм. Інструменти являються лише допоміжним засобом, тоді як людський фактор визначає результативність роботи.

Другою важливою цінністю є робочий програмний продукт понад вичерпну документацію. Agile не відкидає документацію, однак робить акцент на тому, що саме головним індикатором успіху є функціональна цінність продукту для користувача. Документація повинна бути корисною та достатньою, але не надлишковою.

Третя цінність Agile проголошує співпрацю із замовником понад контрактні умови. На практиці це означає тісний контакт зі стороною, яка замовляє продукт, регулярний обмін інформацією, участь замовника у формуванні вимог та пріоритетів. Agile-команда працює не як виконавець за жорстким контрактом, а як партнер, зацікавлений у створенні максимально корисного і конкурентного продукту.

Четвертою базовою цінністю є готовність до змін понад слідування початковому плану. У традиційних методологіях вважається ризиком і відхиленням зміна вимог, тоді як в Agile вона розглядається як природна частина процесу розробки. Команда має бути здатною швидко адаптуватися до нових умов, коригувати функціонал, змінювати план спринту чи пріоритети у відповідь на зовнішні фактори.

Принципи Agile, сформульовані в маніфесті, конкретизують цінності та визначають практичні орієнтири для команд. Принцип частої доставки

робочого продукту є одним із найважливіших. Agile-команди прагнуть до регулярного випуску невеликих, але цінних частин функціоналу, які можна протестувати, продемонструвати замовнику та отримати зворотний зв’язок. Це дає змогу швидко виявляти помилки, адаптувати вимоги та коригувати напрям розвитку продукту.

Інший принцип наголошує, що зміни вимог є бажаними, навіть на пізніх етапах проєкту. Такий підхід забезпечує високу адаптивність та дає компаніям конкурентну перевагу, адже продукт розвивається відповідно до актуальних ринкових потреб. Agile-команди не бояться змін, а навпаки - використовують їх як шанс покращити кінцевий результат.

Ще одним важливим аспектом є щоденна взаємодія між бізнесом і розробниками. Регулярна комунікація забезпечує прозорість процесу, дозволяє оперативно уточнювати завдання, узгоджувати пріоритети та вирішувати проблеми. Така взаємодія мінімізує ризики непорозумінь, а також сприяє підвищенню довіри між учасниками проєкту.

Agile також підкреслює необхідність створення мотивованих і самоорганізованих команд. Члени команди повинні мати не лише технічні компетенції, але й доволі високий рівень ініціативності, відповідальності та готовності до співпраці. Саме самоорганізація дозволяє команді самостійно визначати способи виконання завдань, оптимізувати процеси та підвищувати ефективність.

Принципи Agile охоплюють також технічні аспекти роботи. Зокрема, вони підкреслюють важливість безперервної уваги до технічної досконалості, рефакторингу, автоматизації тестування та підтримання стабільної архітектури. Це сприяє створенню якісного продукту, який можна швидко розвивати без накопичення технічного боргу [1,2].

Принцип регулярної рефлексії команди є не менш важливим щодо власної роботи. Команда аналізує після кожної ітерації свої процеси, обговорює труднощі та визначає способи їх усунення. Такий підхід сприяє постійному

вдосконаленню, створює культуру відкритості та підвищує продуктивність та взаємної підтримки.

У сукупності цінності та принципи Agile формують філософію, в центрі якої знаходиться командна взаємодія, швидке створення цінності для користувача та здатність до адаптації в умовах невизначеності. Для успішного формування Agile-команди потрібно не лише дотримуватися таких принципів формально, а й інтегрувати їх у корпоративну культуру, стиль управління та організаційні процеси.

1.3. Моделі команд у розробці програмного забезпечення

Розробка програмного забезпечення є складним процесом, що вимагає злагодженої роботи різнопрофільних фахівців. Значною мірою ефективність команди залежить від того, яку вона використовує модель організації роботи. У світовій практиці сформувалося декілька підходів до структурування команд, кожен з яких має свої переваги та сфери застосування. Для Agile-проєктів особливо важливо розуміти, як саме побудована команда, оскільки швидкість взаємодії, рівень самостійності та якість прийняття рішень напряму залежать від структури.

Однією з класичних моделей є функціональна модель команди, в якій учасники згруповані за спеціалізаціями: розробники, тестувальники, дизайнери, аналітики тощо. Такий підхід забезпечує високу глибину експертизи, але часто створює бар’єри в комунікації між групами. У контексті Agile ця модель використовується рідше, оскільки вона ускладнює гнучкість під час виконання задач та швидку взаємодію.

Іншим поширеним варіантом є кросфункціональна команда, яка включає представників різних спеціалізацій, що спільно відповідають за створення готового інкременту продукту. Саме кросфункціональні команди вважаються найбільш ефективними для Agile-проєктів, оскільки забезпечують незалежність від зовнішніх підрозділів та дозволяють досягати результатів швидше.

Ще однією моделлю, що часто використовується в IT-компаніях, є матрична структура, яка передбачає подвійне підпорядкування співробітників: функціональному менеджеру та менеджеру проєкту. Такий підхід забезпечує гнучкість розподілу ресурсів, але може створювати конфлікти інтересів. У межах Agile від матричних структур частково відмовляються, оскільки вони суперечать принципам самоорганізації та прозорості.

Для масштабних організацій характерною є компонентна модель, за якої команди відповідають за окремі частини системи, а саме модулі, сервіси або компоненти. Хоча такий підхід дозволяє зберегти технічну експертизу, він може ускладнювати інтеграцію та тестування продукту. У масштабних Agile-підходах (наприклад, LeSS або SAFe) компонентні команди використовуються тільки у випадках, коли це об’єктивно необхідно.

З огляду на потребу гнучкості, автономності та швидкості доставки цінності, найефективнішою для Agile-проєктів вважається кросфункціональна, самоорганізована команда, у якій у спільний результат кожен учасник робить внесок, а не працює ізольовано у рамках своєї професійної ролі.

Кросфункціональна команда також сприяє підвищенню гнучкості, оскільки дозволяє швидко розподіляти завдання залежно від пріоритетів та поточного завантаження. Завдяки можливості взаємозаміни (хоч і часткової) команда може ефективніше реагувати на зміни вимог і непередбачувані ситуації. Наприклад, частина розробників може допомогти тестувальникам, якщо необхідно терміново виконати тестування нового функціоналу, що суттєво прискорює процес.

З іншого боку, класичні функціональні команди, хоч і забезпечують високу експертність, страждають часто від «ефекту передачі естафети», коли робота над задачами переходить від одного підрозділу до іншого - аналітиків, розробників, тестувальників. Це створює черги, збільшує час виконання та підвищує ризик непорозумінь. Саме тому Agile-команди уникають такої моделі,

віддаючи перевагу інтегрованій структурі, де всі компетенції знаходяться всередині однієї групи.

У контексті сучасних методологій розробки також виділяють stream-aligned (потокові) команди, запропоновані у фреймворку Team Topologies (рис.1.1). Такі команди сфокусовані на конкретному потоці цінності - сервісі або певному напрямі продукту. Stream-aligned команда володіє всіма необхідними компетенціями для підтримки та створення функціоналу, що дозволяє значно зменшити залежність від інших команд.

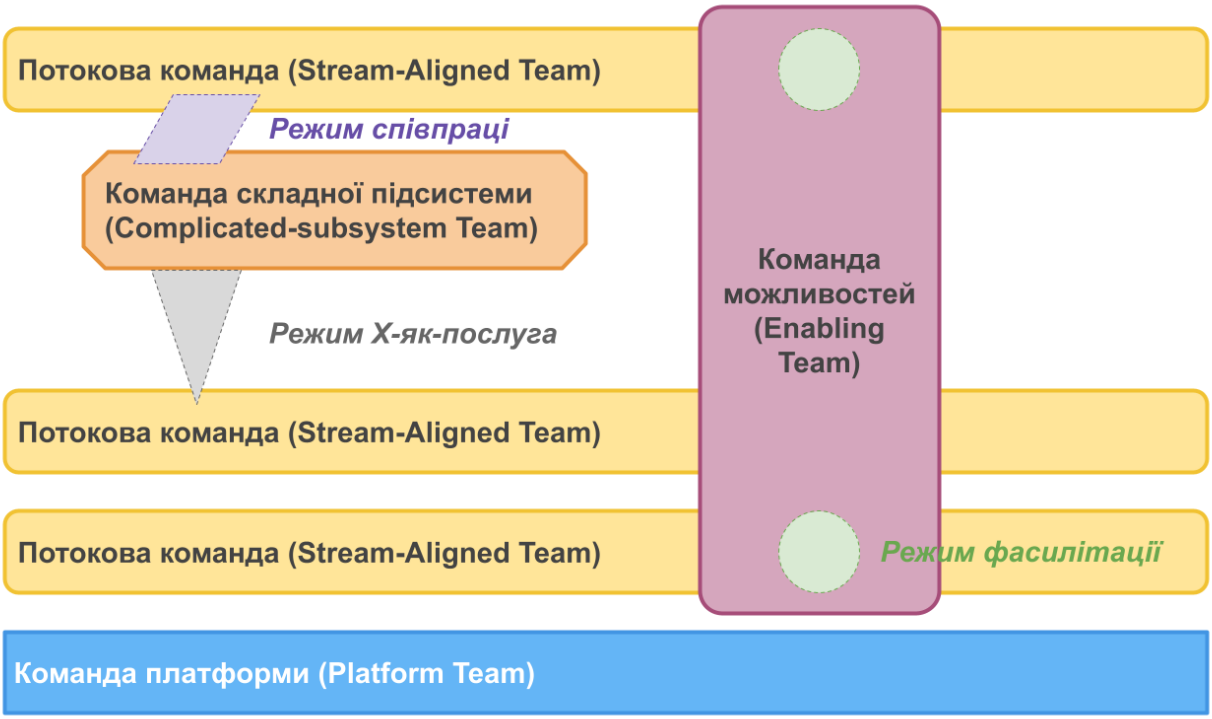


Рис. 1.1 – Структури команд за підходом Team Topologies (stream-aligned, enabling, platform, complicated subsystem)

Enablement-команди інколи застосовують для великих проєктів, які допомагають основним командам упроваджувати нові технічні практики, автоматизацію, DevOps-підходи. Enablement-команди підвищують ефективність інших команд але не створюють продукт, що відповідає принципам Agile про підтримку постійного вдосконалення.

Однією з ключових характеристик Agile-команд є самоорганізація. Це означає, що команда самостійно визначає, як саме виконувати роботу, які технічні рішення обрати, як розподіляти задачі. Самоорганізовані команди демонструють вищу мотивацію, краще розуміння продукту та здатність приймати рішення швидше, ніж це можливо у традиційних ієрархічних структурах [1,3].

Можна стверджувати що у підсумку, сучасні Agile-команди поєднують у собі елементи кросфункціональності, незалежності, самоорганізації, відповідальності за кінцевий результат та високого рівня внутрішньої комунікації. Саме поєднання цих факторів визначає їх здатність ефективно працювати в умовах динамічних проєктів та мінливого ринкового середовища.

1.4. Особливості функціонування Agile-команд

Функціонування Agile-команд ґрунтується на низці принципів, які визначають їхню внутрішню організацію, стиль роботи та спосіб взаємодії між учасниками. На відміну від традиційних команд, що працюють за жорсткими структурами управління, Agile-команди орієнтовані на гнучкість, швидку комунікацію, спільну відповідальність і постійне вдосконалення. Саме ці характеристики забезпечують їх здатність ефективно працювати у складних і динамічних умовах розробки програмного забезпечення.

Однією з ключових особливостей Agile-команд є самоорганізація [3]. Члени команди самостійно визначають, як виконувати роботу, розподіляють завдання між собою, приймають технічні рішення та планують свою діяльність. Це дозволяє команді максимально автономно реагувати на зміни пріоритетів, усувати перешкоди та оптимізувати процеси. Оскільки учасники відчувають свою причетність до результату та відповідальність за нього саме самоорганізація сприяє підвищенню мотивації.

Ще однією характерною рисою є кросфункціональність. Члени Agile-команди мають різні професійні компетенції - від аналізу вимог до

програмування, тестування та підтримки - що дає змогу створювати повноцінний інкремент продукту без залучення зовнішніх підрозділів. Це скорочує час виконання завдань, мінімізує затримки та спрощує комунікацію. Кросфункціональність також допомагає команді зберігати стабільність навіть за умов непередбачуваного завантаження, оскільки учасники можуть взаємно підстраховувати одне одного.

Ефективність Agile-команди багато в чому залежить від якості комунікації. Регулярні зустрічі - щоденні стендапи, планування спринту, ретроспективи, огляди інкрементів - забезпечують прозорість процесів і дозволяють швидко виявляти проблеми. Командна комунікація будується на принципах відкритості, конструктивності та поваги. Відсутність комунікативних бар'єрів сприяє швидкому вирішенню питань і підвищує узгодженість дій (рис.1.2).

Рис. 1.2 – Ключові процеси Scrum-команди: щоденний стендап, планування, ретроспектива, огляд інкременту

Особливої уваги в Agile приділяється інкрементальності та ітеративності. Команда працює короткими циклами — спринтами, після кожного з яких

створюється готовий до використання результат. Це дає можливість замовнику регулярно оцінювати прогрес, а команді — отримувати швидкий зворотний зв’язок і вчасно коригувати напрям роботи. Ітеративність сприяє зменшенню ризиків та підвищенню стабільності процесу [4].

Важливою особливістю Agile-команд є високий рівень прозорості процесів. Усі завдання, статус робіт, пріоритети та проблеми повинні бути видимими для кожного члена команди. Для цього використовуються дошки завдань (наприклад, Kanban або Scrum board), де відображено увесь перебіг роботи: від планування до виконання рис.1.3. Прозорість сприяє відповідальності, дозволяє уникати дублювання задач і забезпечує колективне розуміння поточного стану проєкту.

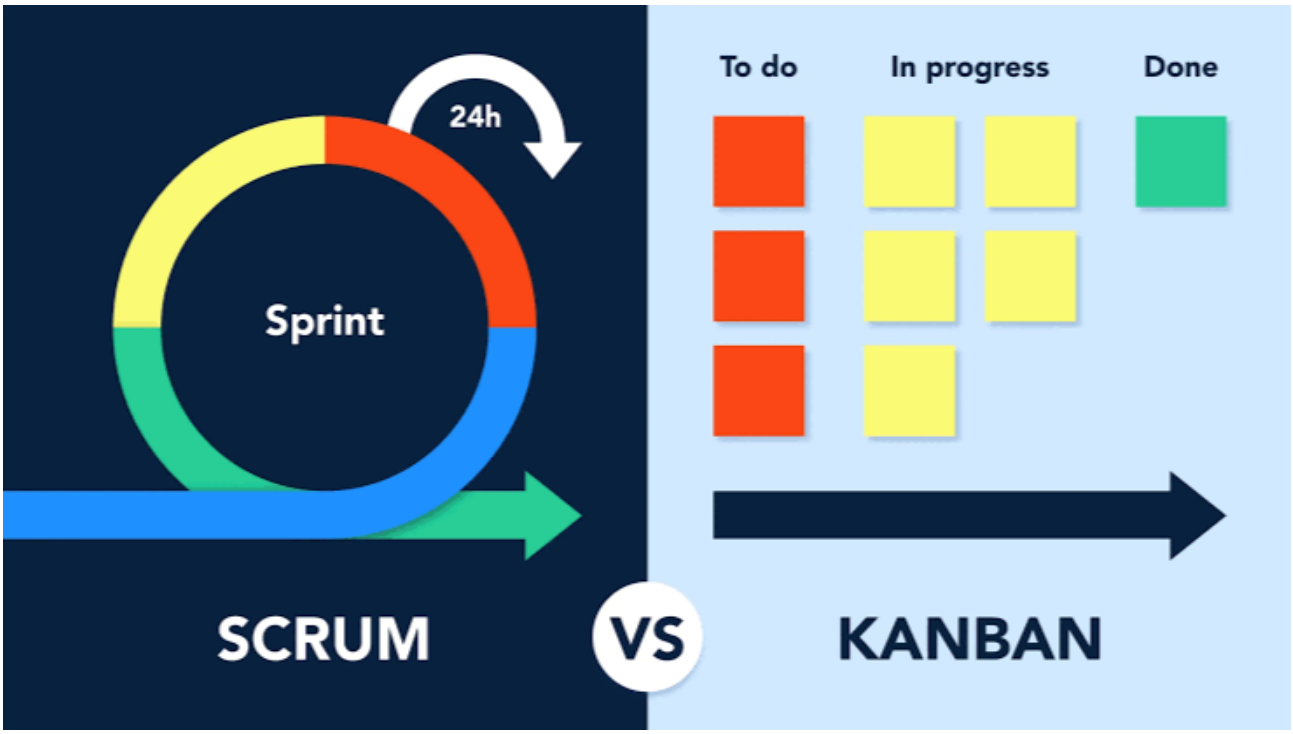


Рис. 1.3 – Візуалізація робочого процесу на Kanban/Scrum-дошці

Ще одним визначальним елементом функціонування Agile-команд є фокус на цінності для користувача. Усі рішення — технічні, організаційні або продуктні — приймаються з урахуванням того, яку користь вони приносять кінцевому споживачеві. Це означає, що команда не просто виконує перелік технічних завдань, а створює функціонал, який має практичну значущість.

Такий підхід забезпечує високу якість продукту і дозволяє швидко реагувати на зміну потреб ринку.

Постійне вдосконалення є ще однією ключовою характеристикою Agile-команд. Рефлексія своєї роботи під час ретроспективи дозволяє обговорювати труднощі, помилки та можливості для покращення. Команда визначає конкретні дії, які потрібно впровадити у наступному спринті, щоб підвищити ефективність та якість роботи. Цей цикл неперервного навчання та вдосконалення формує культуру розвитку й відкритості.

Окремої уваги заслуговує взаємопідтримка та командна відповідальність. У традиційних моделях кожен виконавець відповідає лише за свою частину роботи. В Agile-командах відповідальність за результат спільна, тому всі учасники взаємодіють, допомагають один одному та разом долають труднощі. Це створює сильне відчуття єдності та формує здорову робочу атмосферу, що позитивно впливає на продуктивність [4].

Ще однією особливістю є висока адаптивність. Agile-команда повинна вміти швидко перебудовувати пріоритети, змінювати підхід до задач і впроваджувати нові технології, якщо цього потребує продукт. Адаптивність є важливою у динамічному середовищі, де вимоги можуть змінюватися щотижня, а ринкова ситуація - щодня.

Узагальнюючи, Agile-команди функціонують у середовищі, яке вимагає від них гнучкості, колаборації, відповідальності, відкритості та прагнення до розвитку. Саме поєднання цих факторів забезпечує їхню ефективність і здатність успішно виконувати складні проєкти в умовах високої невизначеності та швидких змін [4].

1.5. Ролі Scrum: Product Owner, Scrum Master, Development Team

Scrum є одним із найпоширеніших фреймворків у межах Agile-методологій і застосовується у тисячах компаній по всьому світу. Його популярність зумовлена простотою, гнучкістю та чітким розподілом ролей, що

забезпечує ефективну організацію роботи команди. У Scrum існує три ключові ролі: Product Owner, Scrum Master та Development Team. Кожна з цих ролей має унікальні функції та зони відповідальності, які взаємодіють між собою для досягнення спільної мети - створення цінного інкременту продукту [4,5].

Роль Product Owner

Product Owner (власник продукту) відповідає за максимізацію цінності продукту, розробленого командою. Це особа, яка формує бачення продукту, взаємодіє зі стейкхолдерами, визначає пріоритети та контролює зміст Product Backlog. Саме Product Owner розуміє бізнес-потреби, аналізує ринок, представляє інтереси замовника й визначає, який функціонал потрібно реалізувати в першу чергу.

Одним із ключових завдань Product Owner є формування та підтримка Product Backlog - списку функціональностей і вимог, впорядкованих за пріоритетом (рис.1.4). Він також відповідає за якість опису задач: вони повинні бути чіткими, зрозумілими та досяжними протягом одного спринту. Продуктова роль потребує високого рівня аналітичного мислення, здатності приймати рішення в умовах невизначеності та ефективного управління пріоритетами.

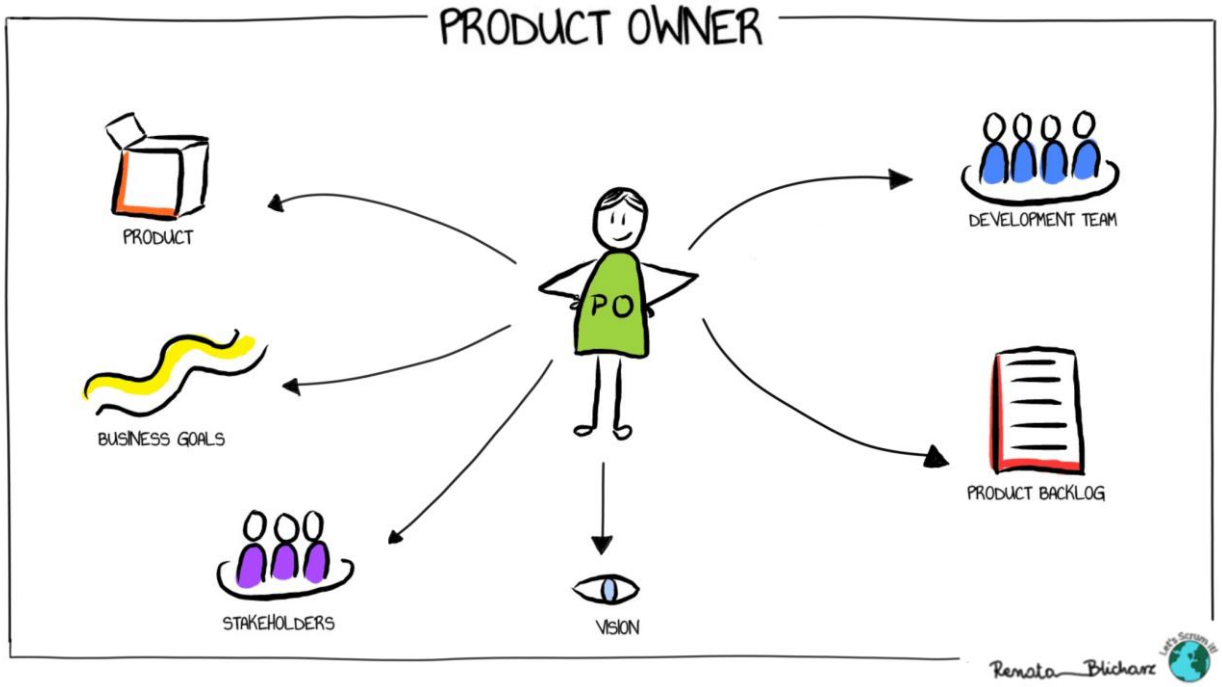


Рис. 1.4 – Основні обов’язки та відповідальність Product Owner у Scrum

Product Owner тісно взаємодіє з командою розробки під час планування спринтів, відповідає на питання щодо вимог та пояснює бізнес-контекст кожної задачі. Він не втручається в технічні рішення, але забезпечує чітке розуміння того, що саме має бути створено і навіщо це потрібно кінцевому користувачеві [5].

Роль Scrum Master

Scrum Master - це ключова роль, яка відповідає за те, щоб Scrum-процес працював ефективно та стабільно. Він не є менеджером чи керівником у традиційному розумінні, а виступає фасилітатором, коучем і захисником команди. Основне завдання Scrum Master - забезпечити дотримання принципів і практик Scrum, а також створити умови, за яких команда може працювати без перешкод рис.1.5.

Scrum Master допомагає команді зрозуміти й імплементувати Scrum-церемонії: планування спринту, щоденні стендапи, огляди інкременту та ретроспективи. Він стежить за тим, щоб ці зустрічі були ефективними, структурованими й орієнтованими на результат. Окрім цього, Scrum Master виявляє та усуває перешкоди, які заважають команді працювати: від організаційних проблем до комунікаційних непорозумінь.

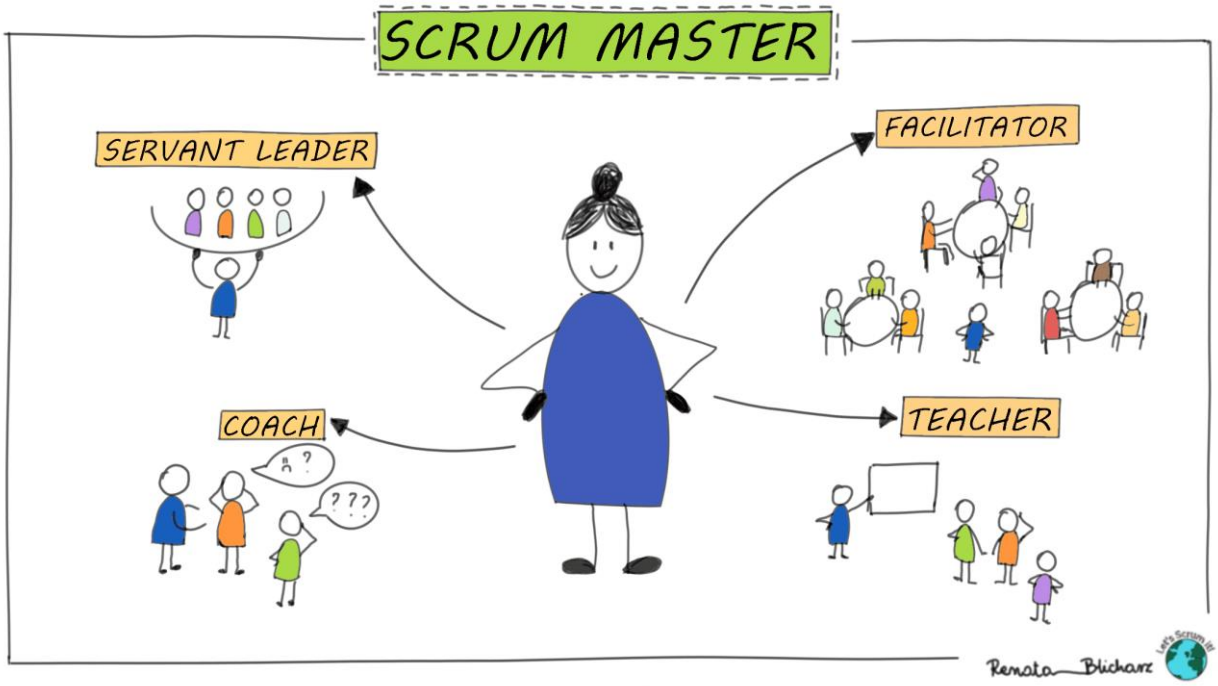


Рис. 1.5 – Роль Scrum Master: фасилітатор, коуч, сервант-лідер

Важливою частиною його роботи є навчання та підтримка команди. Scrum Master допомагає новачкам адаптуватися до процесу, а досвідченим учасникам - вдосконалювати свою роботу. Він також взаємодіє з організацією, пояснюючи керівництву цінність Agile та сприяючи змінам, необхідним для ефективного використання Scrum [4,5].

Роль Development Team

Development Team - це кросфункціональна група фахівців, яка безпосередньо створює інкремент продукту. До її складу зазвичай входять розробники, тестувальники, DevOps-інженери, аналітики та інші спеціалісти, що володіють усіма необхідними компетенціями для повного циклу розробки рис.(1.6).

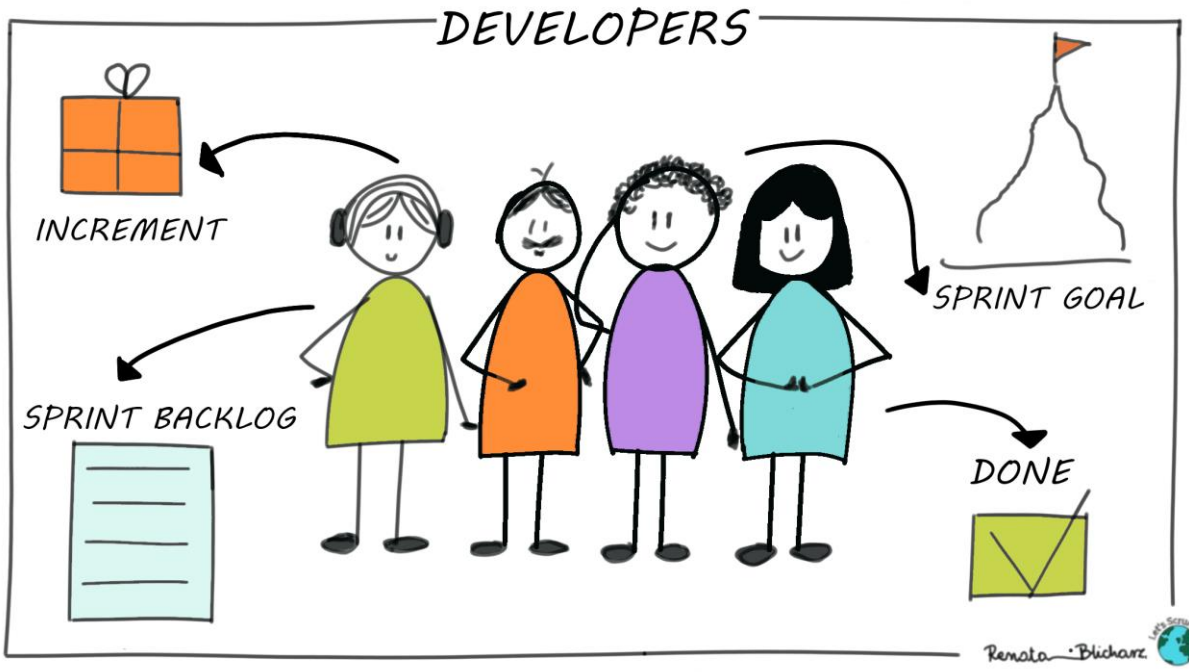


Рис. 1.6 – Структура Development Team та принципи самоорганізації

Команда розробки в Scrum є самоорганізованою, що означає: вона сама вирішує, як виконувати роботу, які технічні підходи застосовувати, як розподіляти задачі та як взаємодіяти всередині групи. Ніхто ззовні не визначає, хто і що має робити — команда бере на себе повну відповідальність за результат.

Характерною рисою Development Team є спільна відповідальність за інкремент. Незалежно від індивідуальних ролей, команда відповідає за якість, функціональність та готовність продукту. Такий підхід формує взаємопідтримку, довіру та високу ефективність [5].

Крім того, Development Team має стабільний склад і постійно вдосконалює власні процеси роботи. У процесі ретроспективи команда визначає, що можна покращити, і впроваджує зміни вже в наступному спринті. Завдяки цьому Scrum-команди здатні підтримувати високу продуктивність навіть в умовах швидких змін.

1.6. Комунікація в Agile-командах

Ефективна комунікація є однією з фундаментальних основ Agile-команд. Саме якісна взаємодія між членами команди, замовниками та стейкхолдерами визначає здатність швидко приймати рішення, своєчасно реагувати на зміни та уникати непорозумінь. На відміну від традиційних підходів, де комунікація часто має формальний та ієрархічний характер, Agile робить акцент на відкритості, прозорості та безперервному обміні інформацією.

Першим важливим аспектом комунікації в Agile є регулярність і ритмічність. Scrum-команди працюють у фіксованих ітераціях — спринтах, кожен з яких має визначений набір зустрічей. Щоденний стендап допомагає синхронізувати роботу, виявити перешкоди та узгодити пріоритети. Планування спринту забезпечує формування єдиних очікувань щодо результатів. Огляд інкременту дозволяє отримати зворотний зв’язок від замовників, а ретроспектива — виявити проблеми та визначити шляхи вдосконалення. Наявність чіткого ритму комунікацій створює передбачуваність і стабільність [5,6].

Другим важливим елементом є прозорість інформації. Усі дані про статус задач, ризики, залежності, пріоритети та цілі команди мають бути доступними для кожного учасника. Це дозволяє уникнути прихованої інформації та затримок, які виникають через неузгодженість. Інструменти типу Jira, Trello чи Azure Boards відіграють значну роль у забезпеченні такої прозорості, надаючи зручні механізми візуалізації прогресу рис.1.5.

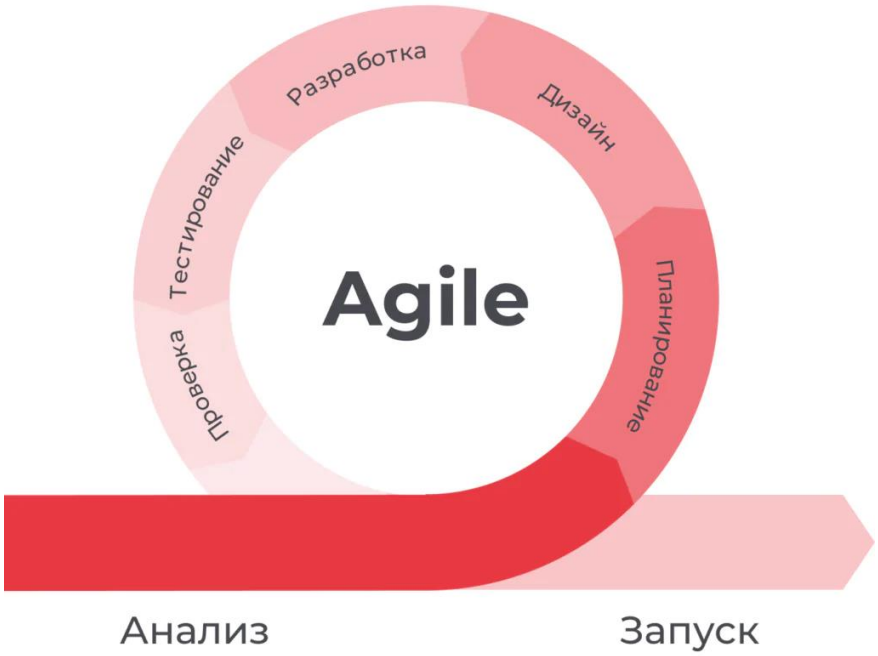


Рис. 1.7 – Візуальний приклад прозорого управління задачами в Agile (Scrum/Kanban board)

Третім аспектом є неформальний характер спілкування. Agile заохочує короткі, прямі діалоги, які допомагають швидко вирішувати питання. У багатьох командах практикується pair programming, відкриті обговорення технічних рішень, спільний аналіз вимог. Це дозволяє уникати бюрократичних погоджень та тривалих листувань [6].

Ще однією важливою складовою ефективної комунікації є спільне розуміння цілей та контексту роботи. Agile-команди працюють не просто над окремими технічними задачами, а над створенням цінності для кінцевого користувача. Тому кожен учасник повинен розуміти бізнес-цілі продукту, логіку його функціонування та очікувану користь. Product Owner відіграє ключову роль у донесенні цього контексту, завдяки забезпеченню спільного бачення та єдину мотивацію для всієї команди [6].

Крім того, комунікація в Agile нерозривно пов’язана з принципом відкритого зворотного зв’язку. Команда повинна бути готовою обговорювати

труднощі, помилки, пропозиції щодо покращення та нові ідеї. На ретроспективах учасники аналізують, що спрацювало добре, що потребує вдосконалення, і розробляють конкретні дії для покращення процесів. Наявність чесного та конструктивного зворотного зв’язку формує здоровий психологічний клімат у команді [4,6].

Суттєву роль відіграє також технічна комунікація, яка стосується вибору архітектурних рішень, підходів до реалізації, практик тестування та автоматизації. У Agile-командах часто застосовуються такі практики, як спільний перегляд коду (code review), обговорення технічних рішень на зустрічах, спільне написання документації. Це дозволяє підтримувати високу якість продукту, сприяє навчанню членів команди одне одного і забезпечення узгодженість технічних підходів.

Окремо слід згадати про важливість довіри та взаємоповаги, які є основою продуктивної комунікації. В Agile-команді кожен учасник має право висловлювати свою думку, ставити запитання та пропонувати рішення. Відсутність страху помилитися або бути розкритикованим сприяє ініціативності та креативності. Одна з ключових передумов високої ефективності команд - психологічна безпека, що підтверджується дослідженнями, зокрема проектом Google “Aristotle”.

Наостанок важливо зазначити, що Agile-комунікація виходить за межі команди та включає взаємодію з зовнішніми стейкхолдерами, бізнесом, користувачами й навіть іншими командами. Від цього залежить узгодженість роботи організації в цілому та можливість забезпечити стабільний розвиток продукту [6] .

Узагальнюючи, можна сказати, що ефективна комунікація в Agile - це складна, багатовимірна система, яка включає відкритість, регулярність, прозорість, взаємоповагу та конструктивність. Вона є одним із визначальних факторів успіху гнучких команд і суттєво впливає на якість продукту, швидкість розробки та задоволеність учасників.

Висновки

У першому розділі було розглянуто фундаментальні поняття, що лежать в основі Agile-методологій та сучасних підходів до формування команд розробки програмного забезпечення. Agile постає як гнучка, орієнтована на цінність та адаптивна методологія, що надає перевагу ітеративній розробці, тісній співпраці із замовником та пріоритету людей над процесами. Комунікація - його ключові цінності, готовність до змін, фокус на результаті та створення робочого продукту - визначають спосіб функціонування команд у сучасному цифровому середовищі.

У межах розділу було розкрито характеристики кросфункціональних та самоорганізованих команд, що становлять основу ефективної роботи в Agile-проєктах. Такі команди здатні швидко адаптуватися до змін, забезпечувати високу якість продукту та приймати спільні рішення завдяки тісній взаємодії між учасниками. Було також проаналізовано різні моделі команд, включаючи структури Team Topologies, які дозволяють оптимізувати потоки цінності та мінімізувати залежності.

Особливу увагу приділено ролям Scrum - Product Owner, Scrum Master та Development Team. Розмежування відповідальностей між цими ролями створює чітку структуру роботи, де кожен учасник команди розуміє свої функції, але водночас бере участь у формуванні кінцевого результату та колективному прийнятті рішень. Product Owner відповідає за бачення продукту, Scrum Master - за ефективність процесу, а Development Team - за технічну реалізацію інкременту.

Комунікація в Agile-командах була визначена як один із ключових факторів успіху. Регулярні зустрічі, прозорість інформації, відкритість до зворотного зв’язку та психологічна безпека створюють передумови для взаємної довіри та високої продуктивності. Agile-команди будують свою взаємодію на принципах поваги, чесності та спільної відповідальності, що сприяє формуванню сприятливого робочого середовища та сталого розвитку.

Підсумовуючи, можна сказати, що Agile-команди поєднують у собі гнучкість, професійність і здатність до постійного навчання, що робить їх особливо ефективними в умовах швидких змін і високої конкуренції на ринку програмного забезпечення. Розуміння цих принципів формує фундамент для подальшого дослідження процесів управління командами та застосування Agile у практичних умовах, що розглядатиметься у наступних розділах дипломної роботи.

РОЗДІЛ 2. МЕТОДОЛОГІЇ УПРАВЛІННЯ КОМАНДАМИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Огляд сучасних підходів до управління командами у сфері IT

Сучасна індустрія розробки програмного забезпечення характеризується високою динамічністю, постійним зростанням вимог та швидкими технологічними змінами з боку користувачів і бізнесу. У таких умовах традиційні підходи до управління командами вже не забезпечують належної ефективності, що зумовлює перехід до адаптивних моделей організації роботи. Управління командами у сфері IT поступово зміщується від ієрархічних структур до більш гнучких, само організованих та автономних моделей [8].

Одним із ключових принципів сучасного управління стає орієнтація на команду, а не на індивідуальних виконавців. Команда розглядається як цілісна одиниця, здатна відповідати за результат та самостійно приймати рішення. Такий підхід напряму сприяє підвищенню відповідальності, мотивації та залученості учасників, що позитивно впливає на продуктивність.

Важливу роль відіграє і адаптивність процесів, що дозволяє реагувати на зміни бізнес-вимог, зовнішніх факторів або технологій. Гнучкі методології, такі як Agile, Scrum, Kanban, Lean та XP, забезпечують можливість дуже швидко перебудовувати робочі процеси, оптимізувати ресурсне навантаження та змінювати пріоритети. У сучасних IT-компаніях дедалі частіше застосовують гібридні моделі, поєднуючи різні методології залежно від специфіки продукту або проєкт у [8].

Суттєвим аспектом сучасного управління командами є мультидисциплінарність. Команди формуються таким чином, щоб володіти всіма необхідними компетенціями для створення повноцінного інкременту продукту. Це дозволяє зменшити залежність від зовнішніх підрозділів, прискорити розробку та підвищити результативність.

Важливою тенденцією також є розвиток культури співпраці. Успішні IT-команди працюють у середовищі, де цінується відкритість, обмін знаннями та колективне прийняття рішень. Комунікація стає ключовою умовою успішної роботи, забезпечуючи синхронізацію дій та прозорість процесів.

У сучасних організаціях спостерігається також зростання ролі лідерства нового типу - сервант-лідерства, зачасту орієнтованого не на контроль, а на підтримку команди. Коучингова модель управління сприяє розвитку автономії та відповідальності і тим самим являється стандартом для Agile-команд.

2.2. Порівняльний аналіз традиційних і гнучких методологій управління командою

Традиційні методології управління проєктами, такі як Waterfall або каскадна модель, довгий час були домінуючими в індустрії розробки програмного забезпечення. Вони ґрунтуються на детальному попередньому плануванні, суворій фіксації вимог та послідовному виконанні етапів - від аналізу до тестування й впровадження. Такий підхід дуже добре працює у проєктах зі низьким рівнем невизначеності та стабільними вимогами, але в умовах сучасного IT він часто виявляється недостатньо гнучким [9].

Основною проблемою традиційних методологій є жорсткість процесів. Будь-яка зміна вимог потребує значних коригувань у документації, ресурсах і планах, що збільшує навантаження на команду й затримує терміни. Комунікація найчастіше відбувається нерегулярно та формально, а взаємодія між аналітиками, розробниками й тестувальниками має характер передачі естафети. Це створює ризики непорозумінь та помилок.

Гнучкі методології, навпаки, орієнтовані на постійне вдосконалення та адаптацію. Agile-команди працюють короткими ітераціями й регулярно демонструють замовнику інкременти продукту, що забезпечує прозорість і дає можливість оперативно змінювати пріоритети. Основний акцент робиться на створенні цінності для користувача, а не на документації.

Однією з найважливіших відмінностей між гнучкими та традиційними підходами є структура команди. У Agile-команді учасники працюють разом, мають різні компетенції й колективно відповідають за результат. У Waterfall-команда часто поділена на незалежні функціональні підрозділи програмісти, аналітики, тестувальники. У результаті взаємодія має послідовний характер. Це підвищує швидкість реакції та якість роботи (рис. 2.1).

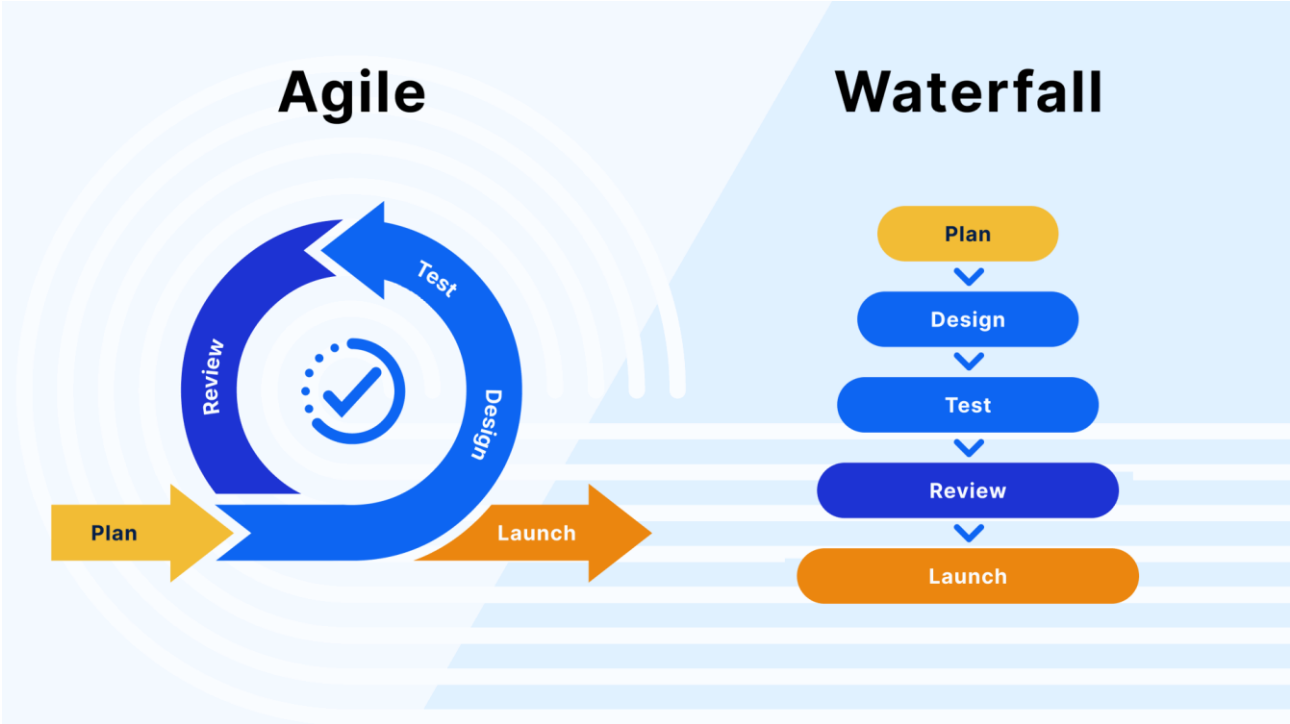


Рис. 2.1 – Порівняння структур команд у Waterfall та Agile (функціональна vs кросфункціональна модель)

Ще однією відмінністю є рівень залучення замовника. У традиційних моделях він зазвичай бере участь лише на початку та наприкінці проєкту. Agile же передбачає постійну взаємодію, спільне формування вимог і швидкий зворотний зв'язок, що дозволяє створювати продукт, максимально відповідний очікуванням користувачів.

Гнучкі методології також демонструють значно вищу стійкість до ризиків. Оскільки робота ведеться ітеративно, команда отримує можливість регулярно тестувати гіпотези, перевіряти працездатність рішень і коригувати підхід. У традиційних моделях ризики накопичуються протягом усього циклу

розробки й можуть бути виявлені лише наприкінці, коли зміни вже коштують значно дорожче. Саме тому в умовах невизначеності та швидкозмінних вимог Agile працює ефективніше.

Ще однією відмінністю є механізми комунікації та прийняття рішень. У Waterfall-проєктах рішення приймаються керівництвом, а команда виконує поставлені завдання відповідно до плану. Agile, навпаки, базується на принципах самоорганізації: команда самостійно оптимізує процеси, визначає технічні рішення та планує роботу. Це знижує потребу у мікроменеджменті та підвищує відповідальність кожного учасника [8,9].

З погляду якості продукту значно ефективніші механізми її забезпечення пропонують гнучкі методології. Завдяки регулярним демонстраціям інкременту, безперервному тестуванню та постійному зворотному зв’язку команди можуть запобігати помилкам ще на ранніх етапах розробки. Традиційні методи нерідко виявляють дефекти вже на етапі фінального тестування, що ускладнює виправлення і збільшує витрати.

Особливе значення у сучасних методологіях має культура взаємодії. Agile-команди працюють у середовищі відкритості, підтримки та взаємного навчання, де кожен учасник може впливати на процес прийняття рішень. У Waterfall цю роль частіше виконують менеджери, а члени команди не завжди залучені до стратегічних аспектів розробки. Таким чином, Agile створює умови для розвитку командної автономії та самостійності.

Загалом порівняльний аналіз показує, що гнучкі методології більш ефективні в умовах високої складності, інноваційності та непередбачуваності, тоді як традиційні підходи залишаються актуальними в стабільних, чітко структурованих проєктах із мінімальним ризиком зміни вимог. Отже, від специфіки продукту, масштабу проєкту та організаційного контексту залежить вибір підходу.

2.3. Scrum як домінуюча методологія управління командами в Agile-середовищі

Scrum є однією з найпоширеніших та найефективніших методологій гнучкої розробки програмного забезпечення. Його популярність пояснюється простотою правил, чіткістю ролей та високою адаптивністю до змін. На відміну від традиційних моделей, Scrum не намагається детально визначити кожен аспект розробки, а зосереджується на створенні умов, у яких команда може працювати самостійно приймаючи рішення, максимально ефективно й адаптуючись до змінних вимог [4,7] .

У центрі Scrum лежить поняття ітеративної розробки, що реалізується через спринти - короткі, зазвичай двотижневі цикли роботи, наприкінці яких команда створює готовий до використання інкремент продукту. Такий підхід дозволяє замовнику регулярно оцінювати прогрес, надавати зворотний зв’язок та вносити корективи. Команда ж може швидко реагувати на зміни й оптимізувати свою роботу, не створюючи надмірної документації (рис.2.2).

Scrum також визначає чітку систему церемоній, що структурують процес розробки [10]. Кожна церемонія виконує свою функцію: планування спринту визначає обсяг роботи; щоденний стендап забезпечує синхронізацію та обмін інформацією; огляд інкременту дає змогу перевірити результати; ретроспектива - виявити можливості для вдосконалення. Завдяки такій структурі команда працює передбачувано та ритмічно.

Рис. 2.2 – Структура Scrum-процесу: основні церемонії, спринт та інкремент

Не менш важливою є роль артефактів Scrum, до яких належать Product Backlog, Sprint Backlog та інкремент. Ці артефакти забезпечують прозорість, контроль та чітке розуміння пріоритетів як для команди, так і для стейкхолдерів. Вони допомагають уникнути хаотичності, яку часто помилково приписують гнучким підходам [4,7] .

Scrum також активно підтримує принципи самоорганізації та кросфункціональності. Команда самостійно визначає, як досягти цілей спринту, а учасники володіють різними компетенціями, що дозволяє створювати повноцінний інкремент без зовнішній залежностей. Це забезпечує високу продуктивність і швидкість реакції на зміни.

Додатковою перевагою Scrum є його здатність масштабуватися для великих організацій і складних багатокомандних проєктів. Хоча базовий Scrum призначений для однієї команди чисельністю до дев’яти осіб, існують розширені фреймворки, такі як Nexus, LeSS або SAFe, які дозволяють координувати роботу десятків команд. Такі підходи зберігають принципи

Scrum, забезпечуючи водночас синхронізацію та узгодженість між командами (рис.2.3).

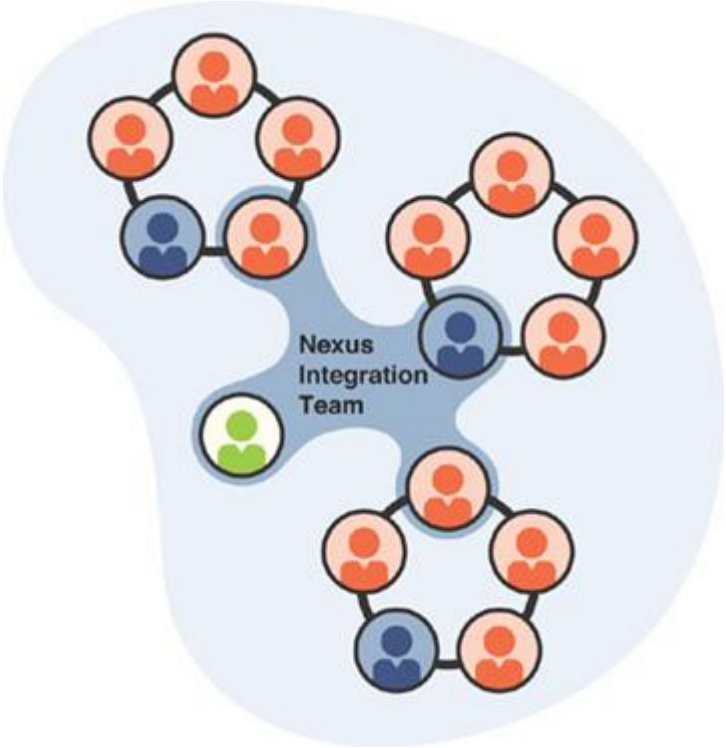


Рис. 2.3 – Масштабування Scrum: приклад взаємодії багатьох команд у рамках Nexus/LeSS

Scrum також активно підтримує культуру прозорості. Усі результати роботи, плани та очікування є відкритими для членів команди та стейкхолдерів. Це сприяє колективному прийняттю рішень, запобігає інформаційним бар'єрам і забезпечує ясність цілей. Регулярні церемонії, огляд інкременту та ретроспектива - дозволяють не лише оцінювати результати, але й формувати спільну відповідальність за процес і кінцевий продукт [4,11] .

Ще одним критично важливим елементом Scrum є орієнтація на цінність, а не на обсяг виконаних завдань. Команда фокусується на тому, щоб створити інкремент, який приносить максимальну користь для користувача або бізнесу. Цей підхід мотивує учасників обговорювати не лише технічні аспекти, а й

бізнес-контекст, що дозволяє уникати розробки непотрібного функціоналу і підвищує загальну якість рішень.

Scrum-команди також приділяють значну увагу якісній комунікації. Щоденна взаємодія, відкритість до зворотного зв'язку, готовність обговорювати складні питання та підтримка психологічної безпеки створюють сприятливе робоче середовище. Це не лише покращує продуктивність, а й сприяє професійному зростанню учасників [11] .

Крім цього, Scrum методично підтримує цикл постійного вдосконалення. Жоден спринт не завершується без ретроспективи - зустрічі, де команда аналізує помилки, обговорює успіхи та визначає конкретні кроки для покращення процесу. Scrum-команди стають більш зрілими, ефективними та стійкими до змін.

Усі ці характеристики роблять Scrum одним із найпотужніших інструментів управління командами в сучасному Agile-середовищі. Scrum дозволяє створювати високоякісні продукти, підтримуючи стабільний розвиток команди та сприяючи її професійній еволюції завдяки ясності ролей, регулярному ритму роботи, прозорості та орієнтації на цінність [4,7].

2.4. Kanban як метод оптимізації робочих процесів у команді

Цей підхід широко застосовується як у продуктових командах, так і в операційних підрозділах, службах підтримки чи аналітичних групах. Kanban є ще одним популярним підходом у межах Agile-методологій, який зосереджується на оптимізації потоку роботи та підвищенні ефективності команд. На відміну від Scrum, що працює у фіксованих ітераціях, Kanban передбачає безперервний потік задач, що дозволяє швидше реагувати на зміни й мінімізувати простої.

Основою Kanban є візуалізація робочого процес (рис.2.4). Для цього використовується Kanban-дошка, на якій задачі відображаються у вигляді карток, що переміщуються між колонками - від «Заплановано» до «Готово».

Така візуальна модель дозволяє команді швидко оцінити поточний стан проєкту, уникнути перевантаження учасників та виявити проблемні місця. Візуалізація робить процес прозорим і полегшує комунікацію навіть у великих командах [6, 12].

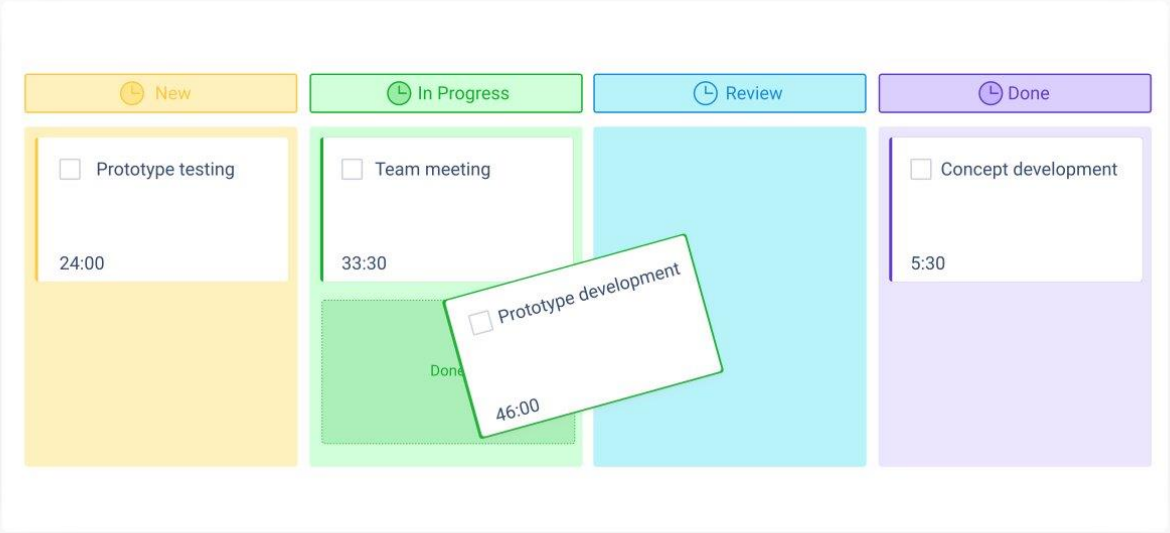


Рис. 2.4 – Приклад Kanban-дошки з колонками «To Do», «In Progress», «Review», «Done»

Однією з ключових практик Kanban є обмеження WIP (Work in Progress) - кількості задач, які команда може виконувати одночасно. Це дозволяє уникнути ситуацій, коли виконавці беруть надто багато роботи, що призводить до зниження продуктивності й затримок. Обмеження WIP сприяє зосередженості та стабільному потоку роботи.

Kanban також підтримує принципи безперервного вдосконалення. Команда регулярно аналізує свої процеси, шукає вузькі місця, оптимізує розподіл задач і адаптує робочий процес під поточні потреби. Це робить Kanban гнучким інструментом, що легко інтегрується в різні типи команд та проєктів [6].

Ще однією суттєвою перевагою Kanban є гнучкість у плануванні. На відміну від Scrum, де команда бере на себе фіксоване зобов’язання на спринт,

тут не має жорстко визначених дедлайнів або ітерацій. Завдання можуть додаватися до потоку роботи в будь-який момент, а пріоритети легко коригуються залежно від актуальних потреб продукту чи бізнесу. У середовищах де неможливо спрогнозувати обсяг роботи наперед - наприклад, у службі технічної підтримки або при роботі з інцидентами, це робить Kanban особливо зручним.

Також важливою характеристикою Kanban є фокус на стабільності потоку. Команда вимірює ключові метрики - такі як час циклу (cycle time), швидкість потоку (throughput) та час очікування (lead time) - щоб зрозуміти, наскільки стабільно й ефективно рухаються задачі. Метою є створення передбачуваного, рівномірного процесу, який дозволяє робити точніші прогнози та мінімізувати затримки [6].

Kanban заохочує прозорість і командну відповідальність, подібно до інших Agile-підходів. Учасники щодня бачать реальний стан роботи, можуть оцінити завантаження команди та виявити вузькі місця. Це сприяє відкритим обговоренням, швидкому вирішенню проблем і конструктивній співпраці.

Цей підхід також легко комбінується з іншими методологіями. Багато команд використовують Scrumban - гібрид Scrum і Kanban, який об’єднує структуру Scrum-ітерацій із гнучкістю безперервного потоку Kanban. Це дає можливість адаптувати процес під унікальні потреби команди й продукту.

Kanban добре підходить для команд будь-якого розміру: від невеликих продуктових груп до великих відділів розробки. Завдяки простоті впровадження та широким можливостям для адаптації Kanban став універсальним інструментом для оптимізації процесів у різних сферах IT.

Узагальнюючи, Kanban - це не просто візуальна дошка, а повноцінна система управління потоком робіт, яка дозволяє зменшити хаос, підвищити продуктивність, зменшити кількість незавершених задач та забезпечити стабільну швидкість розвитку продукту. Kanban стає незамінною частиною

інструментарію Agile для команд, які прагнуть покращити прозорість і ефективність.

2.5. Lean-підходи у побудові ефективних команд

Lean — це підхід до організації роботи, що походить від виробничої системи Toyota, але з часом був адаптований до IT-сфери та розробки програмного забезпечення [5,14] . Основна мета Lean — усунення всіх видів втрат, оптимізація процесів і створення максимальної цінності для користувача за мінімальних витрат часу та ресурсів. Lean і Agile мають багато спільного, однак Lean робить більший акцент саме на ефективності потоку, якості та усуненні зайвих дій (рис.2.5).

Центральним поняттям Lean є цінність (value) — те, за що користувач готовий платити або що приносить йому безпосередню користь. Усі процеси, що не створюють цінність, вважаються втратами й повинні бути або оптимізовані, або навіть повністю усунені. У контексті розробки програмного забезпечення такими втратами можуть бути надлишкова документація, багатоступенева бюрократія, зайві наради або довгі цикли погодження.

Lean визначає сім основних типів втрат, що мають найбільший вплив на продуктивність команд: зайві переміщення, надлишкове виробництво, дефекти, очікування, надлишкова обробка, зайва інвентаризація та нераціональне використання талантів. Усі ці фактори, адаптовані до IT, проявляються як затримки у виконанні задач, дублювання роботи, баги, недокомунікація та неефективне використання компетенцій учасників [5,14].

Рис. 2.5 – Сім типів втрат Lean, адаптовані для розробки програмного забезпечення

Однією з ключових практик Lean є Kaizen - культура постійного вдосконалення, яка передбачає регулярний аналіз процесів та впровадження невеликих, але систематичних покращень. Команди, які працюють за Lean-підходом, розвивають навичку швидкого виявлення проблем і пропонування рішень, що дозволяє суттєво зменшувати втрати й підвищувати ефективність [5,14].

Ще одним ключовим принципом Lean є створення безперервного та стабільного потоку роботи. Це означає, що команда повинна мінімізувати процесси сповільнюють просування задач а саме - затримки, вузькі місця та переривання. Для цього застосовуються такі техніки, як стандартизація процесів, оптимізація черг задач та вирівнювання навантаження між учасниками. У результаті команда працює більш передбачувано, з меншими коливаннями продуктивності та якіснішими результатами [5].

Lean також підкреслює важливість визуалізації процесів, що дозволяє краще розуміти потік робіт і швидко виявляти проблемні зони. У багатьох Lean-командах використовуються аналоги Kanban-дошок, де відображається статус

задач, часові затримки та критичні точки. Це дозволяє команді колективно аналізувати процес і приймати рішення, що спрямовані на його покращення.

Важливим аспектом Lean є повага до людей - принцип, що визнаний одним із фундаментальних у філософії Toyota. У контексті IT це означає створення умов, за яких кожен член команди може максимально використовувати свої компетенції, пропонувати нові рішення, впливати на покращення процесів і відчувати свою цінність. Lean-команди формують середовище відкритості та довіри, де заохочується ініціативність і відповідальність.

Lean також підтримує концепцію мінімально життєздатного продукту (MVP), яка дозволяє швидко перевірити гіпотези та отримати реакцію користувачів. MVP допомагає команді уникати розробки зайвого функціоналу, зосереджуючись на ключових можливостях, які створюють основну цінність. Це прискорює цикл зворотного зв'язку та забезпечує кращу відповідність продукту реальним потребам [5,15].

Крім того, усунення варіативності та нестабільності, яка може виникати в роботі команд через непередбачуваність обсягів задач, неузгодженість процесів або зовнішні залежності це те на що спрямований Lean. Суттєво зменшити вплив таких факторів і зробити роботу більш плавною дозволяє використання стандартних робочих підходів, автоматизація повторюваних дій та узгодженість процесів.

Узагальнюючи, Lean-підходи формують культуру ефективності, безперервного вдосконалення та командної відповідальності. Lean допомагає усувати втрати, підвищувати якість, оптимізувати потік робіт і створювати цінність швидше та ефективніше. Саме тому Lean є важливою частиною сучасних Agile-команд, доповнюючи Scrum, Kanban та інші методології, і сприяє формуванню високопродуктивних і зрілих колективів [5,15].

2.6. Підходи до масштабування Agile у великих командах

Agile-підходи були спочатку створені для невеликих команд чисельністю до 7–9 осіб. Проте розвиток IT-індустрії, збільшення складності продуктів і потреба координувати роботу десятків або навіть сотень фахівців спонукали до створення підходів, що дозволяють масштабувати Agile на рівень великих організацій. Масштабування Agile не означає просте копіювання практик Scrum або Kanban, а вимагає створення системи координації, синхронізації та взаємодії між багатьма командами, при цьому зберігаючи гнучкість, автономність і орієнтацію на цінність [16].

Однією з ключових проблем масштабування є складність комунікації. Коли в розробці задіяно десятки команд, виникають ризики дублювання роботи, суперечностей у технічних рішеннях, затримок через залежності та інші координаційні труднощі. Тому методи масштабування пропонують структури, які забезпечують синхронізацію команд, прозорість пріоритетів і погодженість інкрементів продукту.

Сучасні підходи до масштабування Agile можна умовно поділити на три основні групи: фреймворки, орієнтовані на мінімальне втручання (наприклад, Nexus), методи з глибокою організаційною трансформацією (SAFe), а також підходи, спрямовані на зменшення складності (LeSS). Вибір моделі масштабування залежить від розміру компанії, складності продукту та рівня зрілості команд.

Nexus — це легкий фреймворк для масштабування Scrum на рівні 3–9 команд, який фокусується на управлінні залежностями та інтеграцією інкрементів. Він зберігає більшість практик Scrum, доповнюючи їх ролями та артефактами для багатокомандної взаємодії [16] .

Ще одним із найвідоміших підходів є SAFe (Scaled Agile Framework) — комплексна система масштабування Agile, яка охоплює всі рівні організації: командний, програмний, портфельний та бізнес-рівень. SAFe пропонує структуру з чіткими ролями, практиками, артефактами й церемоніями, що

дозволяють узгоджувати роботу великої кількості команд у межах одного продукту або навіть кількох напрямів.

Ключовою особливістю SAFe є Agile Release Train (ART) — «поїзд», що об’єднує 8–12 команд, які рухаються синхронно, виконуючи роботу в межах загальних часових рамок, завдяки чому забезпечується узгоджений реліз функціональності. SAFe широко впроваджується великими корпораціями, які потребують масштабування Agile без втрати передбачуваності та стабільності.

На відміну від SAFe, підхід LeSS (Large-Scale Scrum) є мінімалістичним і простішим. Він зберігає максимальну чистоту Scrum, застосовуючи його принципи до великої кількості команд. LeSS наголошує на зменшенні організаційної складності, усуненні зайвих ролей і фокусі на спільному продукті. Команди у LeSS працюють у межах одного Product Backlog, мають єдиного Product Owner та спільне бачення. Такий підхід мінімізує бюрократію, залишаючи практики Scrum в основі масштабування [6,16] .

Ще одним підходом, який здобув популярність завдяки Spotify, є Spotify Model - гнучка культура, побудована навколо автономних команд («скводів»), які об’єднуються у більші структури (трайби, чаптери й гільдії). Spotify Model не є формальною методологією, а радше філософією побудови гнучкої та інноваційної організації, де команди мають високий рівень автономії, а взаємодія між ними забезпечується через синхронізаційні групи.

Основна ідея Spotify-підходу - створення середовища, яке стимулює інновації, творчість і відповідальність команд, а не жорстку регламентацію процесів. Завдяки цьому Spotify Model часто використовується компаніями, які прагнуть гнучкості та швидкого розвитку, але не потребують жорстких структур SAFe [17].

Кожен із підходів має свої переваги й недоліки. SAFe забезпечує масштабованість і контроль, але може бути громіздким. LeSS — легкий і гнучкий, проте підходить не всім організаціям. Spotify Model пропонує високу

автономію, але потребує зрілої культури. Вибір моделі залежить від контексту, структури компанії та рівня зрілості команд [17].

Завершальним аспектом масштабування Agile є забезпечення узгодженості між технічними рішеннями різних команд. У великих організаціях це одна з найскладніших задач, оскільки різні команди можуть використовувати різні технології, підходи до архітектури чи стандарти якості. Тому фреймворки масштабування передбачають механізми синхронізації: архітектурні ради, технічні перегляди, спільні гілки розробки або регулярні зустрічі інженерних лідерів. Ці механізми дозволяють уникати конфліктів між інкрементами та забезпечують цілісність продукту.

Важливим викликом масштабування є також управління залежностями. Чим більше команд бере участь у проєкті, тим більшою стає ймовірність того, що робота однієї команди блокує іншу. Для мінімізації таких ризиків фреймворки пропонують різні техніки: декомпозицію продукту на незалежні модулі, узгоджені пріоритети між командами, регулярну інтеграцію та спільні релізні цикли. У Nexus та LeSS, наприклад, велику увагу приділено інтегрованому інкременту, який створюється спільними зусиллями всіх команд.

Також ключовим елементом масштабування Agile є підтримка прозорості на всіх рівнях організації. Це включає спільні візуальні системи управління, такі як багаторівневі Kanban-дошки, загальні дорожні карти продукту, синхронізовані беклоги та відкриті метрики ефективності. Прозорість допомагає учасникам швидко розуміти ситуацію, приймати рішення та координувати роботу між командами без надмірного менеджерського контролю [6,17].

Ще одним важливим фактором є культура організації. Масштабування Agile неможливе без підтримки керівництва, готовності змінювати традиційні підходи, зменшувати бюрократію та делегувати повноваження командам. Успішні організації переходять від команд, що працюють незалежно, до

«екосистем команд», де кожна група має автономію, але одночасно працює в рамках спільної цілісної стратегії.

Варто також підкреслити роль інструментів автоматизації, які є критично важливими при масштабуванні. CI/CD-пайплайни, автоматичне тестування, централізовані системи логування та моніторингу дозволяють уникнути хаосу, пришвидшити релізи й забезпечити однаковий рівень якості у всіх командах. Без високого рівня автоматизації масштабування Agile стає майже неможливим.

Підсумовуючи, масштабування Agile - це комплексний процес, що потребує одночасної адаптації структури, процесів, культури та технічних практик. Успіх залежить від здатності організації створити баланс між автономією команд і центральною координацією, які забезпечують гнучкість і узгодженість на високому рівні. Вибір моделі масштабування має базуватися на потребах конкретного продукту та рівні зрілості організації, а також на готовності до змін [3,18].

Висновки

У другому розділі були систематизовані та проаналізовані основні методології, що застосовуються для управління командами розробки програмного забезпечення у сучасних IT-організаціях. Кожен із розглянутих підходів — Scrum, Kanban, Lean та масштабовані Agile-фреймворки — виконує свою унікальну роль у формуванні ефективного робочого середовища та оптимізації процесу розробки.

Аналіз показав, що Scrum залишається однією з найпоширеніших методологій завдяки своїй структурованості, чіткому розподілу ролей та високому рівню прозорості. Регулярний ритм роботи, церемонії та артефакти Scrum формують стабільний процес, який дозволяє командам швидко адаптуватися до змін і підтримувати високий рівень продуктивності. Scrum також сприяє самоорганізації, що є важливим фактором для розвитку командної автономії.

Kanban, у свою чергу, пропонує більш гнучкий і потокоорієнтований підхід. Він дозволяє командам працювати без жорстких ітерацій і надає інструменти для контролю завантаження, зменшення затримок та оптимізації потоку задач. Kanban є простим у впровадженні й придатним для команд будь-якого типу — від продуктових груп до операційних підрозділів.

Lean-підходи акцентують увагу на усуненні втрат, підвищенні якості та створенні стабільного потоку роботи. Lean формує культуру постійного вдосконалення, що дозволяє командам знижувати варіативність, пришвидшувати розробку й ефективніше використовувати ресурси. Завдяки акценту на цінності для користувача Lean добре доповнює інші Agile-методології.

Окремий акцент у розділі було зроблено на масштабованих підходах Agile, які дають змогу координувати роботу великої кількості команд. Фреймворки SAFe, LeSS, Nexus і Spotify Model пропонують різні механізми синхронізації, управління залежностями та забезпечення спільної цінності. Їх

застосування стає критично важливим у великих організаціях, де необхідно поєднати автономію команд зі спільними стратегічними цілями.

Загалом усі методології, розглянуті у цьому розділі, спрямовані на покращення взаємодії команд, підвищення прозорості, забезпечення стабільності потоку та створення продуктів з максимальною цінністю для користувачів. Вибір конкретного підходу залежить від характеру проєкту, рівня зрілості команди та організаційного контексту.

Таким чином, сучасні методи управління командами дозволяють організаціям досягати більшої ефективності, швидше адаптуватися до умов ринку та забезпечувати високу якість програмного забезпечення, що створює основу для подальшого вивчення процесів формування та розвитку команд у наступному розділі.

РОЗДІЛ 3. ФОРМУВАННЯ КОМАНДИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В AGILE-ПРОЄКТАХ

3.1. Принципи формування високоефективних Agile-команд

Формування команди в Agile-проєктах є одним із найважливіших етапів, що визначає успішність подальшої розробки продукту. На відміну від традиційних моделей, де склад команди часто визначається на основі функціональних ролей, Agile підходить до цього процесу більш комплексно, враховуючи компетенції, здатність до взаємодії, рівень автономії та потенціал розвитку кожного учасника. Високоефективна Agile-команда — це не лише група фахівців, а збалансована система, здатна самоорганізовуватися та швидко адаптуватися до змін [18,19].

Одним із ключових принципів формування команди є кросфункціональність - наявність у команді всіх необхідних компетенцій для створення повноцінного інкременту продукту. Це дозволяє мінімізувати залежності від зовнішніх підрозділів, скоротити час на узгодження та забезпечити швидке просування задач. Кросфункціональні команди можуть включати розробників, тестувальників, аналітиків, дизайнерів, DevOps-фахівців та інших спеціалістів, залежно від специфіки продукту.

Ще одним принципом є самоорганізація, що передбачає здатність команди самостійно визначати спосіб виконання роботи, розподіляти задачі та приймати технічні рішення. Самоорганізовані команди демонструють вищу відповідальність, кращу мотивацію та здатність швидко реагувати на зміни. Цей принцип також знижує потребу в мікроменеджменті та сприяє зростанню професійної зрілості учасників.

Важливою особливістю є баланс компетенцій і ролей. У команді повинні бути як технічні спеціалісти, так і аналітичні й творчі фахівці. Баланс умінь дозволяє не лише покривати всі етапи розробки, а й збагачувати команду різними точками зору, що сприяє кращому прийняттю рішень. Успішні Agile-

команди часто використовують T-shaped model — коли спеціаліст має глибоку експертизу в одній сфері, але володіє базовими навичками в інших.

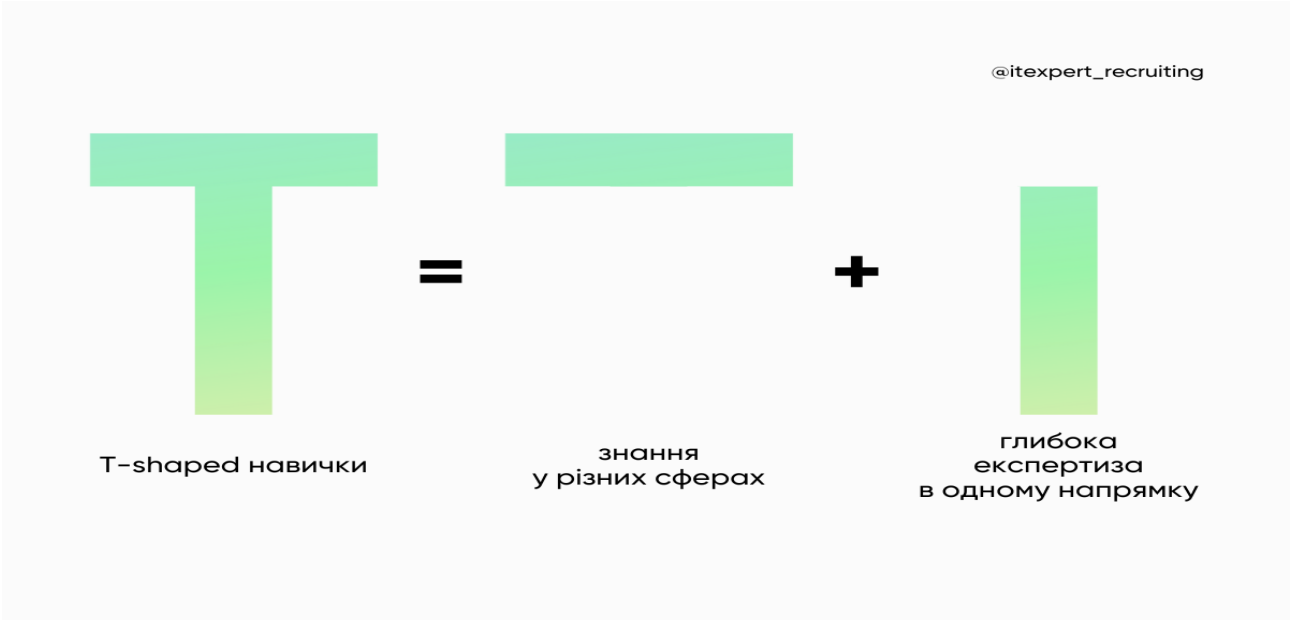


Рис. 3.1 – T-shaped компетенції фахівця в Agile-команді

Ще одним важливим принципом формування Agile-команд є стабільність складу, яка забезпечує поступове зростання ефективності. Дослідження у сфері командної динаміки підтверджують, що стабільні команди демонструють кращу продуктивність завдяки накопиченню спільного досвіду, зростанню взаємної довіри, узгодженості в комунікаціях та формуванню спільних робочих патернів. Часті зміни у складі команди можуть призводити до втрати експертизи, зниження мотивації та порушення командного ритму [20].

У процесі формування команди велике значення має психологічна безпека - відчуття учасників, що вони можуть відкрито висловлювати власні думки, не боячись осуду чи покарання. Дослідження Google Project Aristotle показало, що психологічна безпека є ключовим фактором ефективності команд [20]. Agile-команди, які працюють в атмосфері відкритості, приймають креативніші рішення, швидше виявляють проблеми та активно пропонують інновації. Для створення такої атмосфери Scrum Master або Agile-коуч має стежити за тим, щоб команда працювала у середовищі взаємної підтримки.

Рис. 3.2 – Фактори ефективності команди за результатами дослідження Google Project Aristotle

Не менш важливим принципом є спільна відповідальність за результат. На відміну від традиційних моделей, де кожен виконав певну «свою частину роботи», Agile-команди несуть колективну відповідальність за створення завершеного інкременту продукту. Це стимулює взаємодію, спільне розв'язання проблем, обмін знаннями та взаємну підтримку. Коли команда сприймає результат як спільну мету, зменшується кількість блокерів, прискорюється виконання задач і зростає якість продукту.

Ще одним принципом є максимальна прозорість, яка включає відкритий доступ до інформації про задачі, прогрес, проблеми та пріоритети. Швидко приймати рішення, уникати прихованих ризиків і підтримувати довіру між учасниками дозволяє прозорість. У Scrum це забезпечується завдяки інструментам на кшталт Product Backlog, Sprint Backlog, Definition of Done, щоденним стендапам та оглядам інкрементів [7,21].

Не можна оминути увагою і принцип ітеративності розвитку команди. Agile-команда постійно вдосконалюється завдяки регулярним ретроспективам,

на яких учасники аналізують власну роботу, визначають проблеми та формують дії для покращення процесу. Це створює цикл безперервного зростання і сприяє поступовому підвищенню продуктивності.

Завершальним елементом формування Agile-команди є створення чіткої та спільної цілі, яка визначає напрямок роботи й дозволяє кожному учаснику розуміти свій внесок у загальний результат. Успішні команди завжди мають спільне бачення продукту, чітко сформульовані критерії успіху та узгоджені очікування щодо способів досягнення цих цілей. Наявність такого бачення зменшує розбіжності у пріоритетах та допомагає команді працювати узгоджено навіть в умовах невизначеності.

Ще одним важливим аспектом є синергія та командна динаміка — взаємодія учасників, що формує здатність команди працювати як єдине ціле. Згідно з моделлю Брюса Такмена, команди проходять кілька етапів розвитку: формування, конфліктність (storming), нормування та продуктивна робота (performing). Усвідомлення цих етапів дозволяє Scrum Master або Agile-коучу своєчасно реагувати на конфлікти, сприяти налагодженню взаємодії та підтримувати команду в її природному розвитку.

Важливу роль відіграє також правильне поєднання особистих якостей учасників. Agile-команда повинна складатися з людей з різними стилями мислення, комунікації та підходами до розв’язання проблем. Це створює різноманіття перспектив, сприяє інноваційності та підвищує якість рішень. Для оцінки особистих характеристик часто використовуються моделі Belbin Team Roles, MBTI, DISC тощо, які допомагають визначити сильні сторони учасників і сформувати збалансовану команду.

Ще один критично важливий чинник - наявність лідерства служіння (servant leadership). У контексті Agile така роль зазвичай належить Scrum Master або Agile-коучу, які не керують командою у традиційному розумінні, а створюють умови для її максимально ефективної роботи. Вони знімають перешкоди, підтримують розвиток, допомагають оптимізувати процеси і

захищають команду від зайвого зовнішнього впливу. Servant leadership сприяє формуванню атмосфери довіри та безпеки, що безпосередньо впливає на продуктивність.

Ключовим також є забезпечення ефективної комунікації, яка повинна бути відкритою, регулярною та конструктивною. Agile-команди використовують різні інструменти та практики комунікації - щоденні стендапи, спільні чати, дошки задач, регулярні зустрічі для синхронізації. Від якості комунікацій значною мірою залежить швидкість прийняття рішень, уникнення блокерів і взаєморозуміння між учасниками [22].

Підсумовуючи основні принципи, можна зазначити, що формування високоефективної Agile-команди - це комплексний процес, який охоплює технічні компетенції, психологічні аспекти, культуру взаємодії та лідерство. Лише збалансоване поєднання цих факторів дозволяє створити команду, здатну швидко адаптуватися, приймати якісні рішення та досягати високих результатів у складних проєктах [22].

3.2. Ключові ролі в Agile-команді та їх компетенції

Agile-команда - це самодостатня, кросфункціональна група фахівців, які спільно несуть відповідальність за створення інкременту продукту. На відміну від традиційних моделей управління, Agile не передбачає ієрархічної структури з жорстким розподілом на керівників і підлеглих. Натомість кожна роль має чіткі зони відповідальності, які сприяють автономності команди, швидкому прийняттю рішень та ефективній співпраці. У більшості Agile-проєктів основними ролями є: Product Owner, Scrum Master та команда розробки (Developers) [23,24,25].

Product Owner (Власник продукту)

Product Owner відіграє ключову роль у забезпеченні цінності продукту. Він представляє інтереси замовника, користувачів та бізнесу. Основним артефактом, за який відповідає Product Owner, є Product Backlog -

впорядкований список вимог, задач і покращень, який формує майбутнє продукту.

Ключові компетенції Product Owner включають:

- Стратегічне бачення продукту: здатність формувати довгострокові цілі, визначати ринкові потреби та бачити перспективи розвитку продукту.
- Навички пріоритетизації: уміння визначати, які функціональні можливості є найбільш цінними для користувача та мають бути реалізовані першими.
- Комунікаційні навички: постійна взаємодія зі стейкхолдерами, командою розробки та користувачами потребує чіткої та структурованої комунікації.
- Аналіз вимог: здатність перетворювати бізнес-цілі на конкретні задачі, User Stories та критерії приймання (Acceptance Criteria).
- Розуміння принципів UX та технічних обмежень: хоча Product Owner не зобов'язаний бути розробником, базове розуміння технологій допомагає приймати більш обґрунтовані рішення.

Product Owner є тією людиною, яка визначає «що саме» буде будувати команда, але не втручається у процес «як саме» це буде реалізовано, зберігаючи автономію розробників.

Scrum Master

Scrum Master - це роль, що відповідає за ефективність роботи команди та правильне застосування Agile-практик. Він не є керівником у традиційному сенсі, а виступає фасилітатором, наставником і сервант-лідером, який допомагає команді досягти максимальної продуктивності.

Основні компетенції Scrum Master:

- Фасилітація процесів: організація та підтримка проведення щоденних стендапів, планувань, оглядів і ретроспектив. Scrum Master стежить, щоб зустрічі були корисними, а не формальними.

- Усунення перешкод (impediments): допомога команді у вирішенні проблем, які блокують роботу - від процесних питань до зовнішніх залежностей.
- Підтримка самоорганізації: Scrum Master створює умови, у яких команда сама приймає рішення щодо способів виконання задач.
- Навчання Agile-принципам: розвиток команди через коучинг, проведення воркшопів і підтримку культури прозорості та відповідальності.
- Захист команди: від надмірного тиску менеджменту, неузгоджених запитів та постійних змін пріоритетів.

Scrum Master працює не лише з командою, а й зі стейкхолдерами, навчаючи їх правильно взаємодіяти з командою розробки без мікроменеджменту.

Команда розробки (Developers)

Розробники - це кросфункціональна група фахівців, які безпосередньо створюють інкремент продукту. До складу можуть входити програмісти, тестувальники, DevOps-інженери, дизайнери, аналітики - будь-які спеціалісти, необхідні для завершення задачі «під ключ». Основна особливість цієї ролі - відсутність внутрішньої ієрархії: усі учасники мають рівні права та спільну відповідальність за результат.

Головні компетенції команди розробки:

- Технічна експертиза: володіння інструментами, мовами програмування, фреймворками та підходами, необхідними для створення якісного продукту.
- Взаємодія та командна робота: уміння ефективно комунікувати, підтримувати колег, проводити парне програмування, рев'ю коду та обмінюватися знаннями.
- Оцінювання задач: участь у плануванні та прогнозуванні обсягу роботи, використання методів story points, planning poker тощо.

- Самоорганізація: самостійне розподілення задач, вибір технічних рішень і оптимальних способів реалізації.
- Гарантія якості: написання тестів, автоматизація процесів, дотримання Definition of Done та технічних стандартів.

Усі розробники несуть спільну відповідальність за створення інкременту продукту у межах спринту. Це виключає ситуації, коли «хтось щось зробив, а результат не готовий».

3.3. Моделі взаємодії в Agile-командах

Ефективна взаємодія є фундаментом роботи будь-якої Agile-команди. Оскільки Agile базується на швидкій комунікації, прозорості та спільній відповідальності, моделі взаємодії повинні забезпечувати максимально простий та передбачуваний обмін інформацією між учасниками. У сучасних командах найпоширенішими моделями взаємодії є: Face-to-Face комунікація, візуальні інструменти управління, регулярні синхронізації та цифрові сервісні моделі співпраці [26].

Однією з центральних моделей є щоденна синхронізація (Daily Scrum). Ця коротка зустріч дозволяє команді узгодити прогрес, виявити блокери та спланувати дії на найближчий день. Daily Scrum підтримує прозорість і дає змогу всім учасникам бачити реальний стан роботи.

Не менш важливою є візуалізація задач через Kanban або Scrum-дошки. Вона дозволяє миттєво зрозуміти, на якому етапі перебуває робота, які задачі блокуються, які виконуються та які заплановані. Візуальна модель знижує кількість непорозумінь та зайвих комунікацій [6].

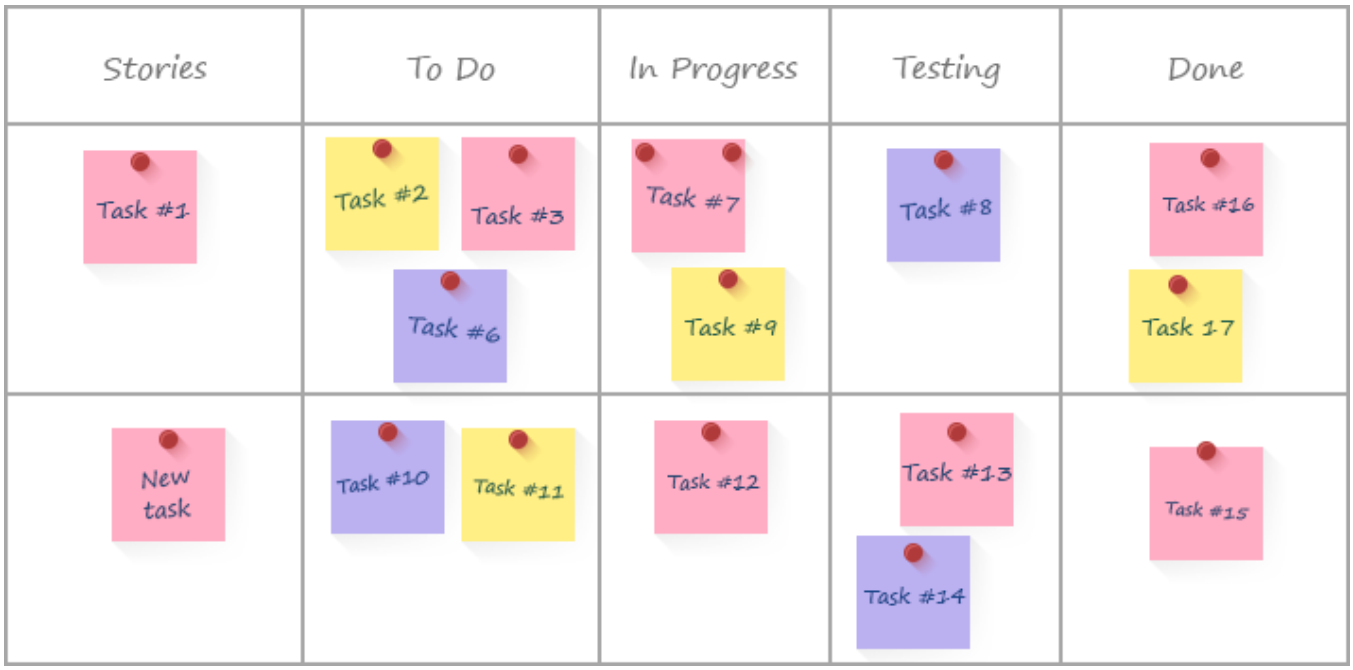


Рис. 3.3 – Scrum/Kanban-дошка як основний інструмент командної взаємодії

У великих командах або в компаніях із розподіленою структурою активно застосовуються цифрові моделі взаємодії: спільні чати, таск-трекери, сервіси для відеозв'язку, документообігу та мозкових штурмів. Такі інструменти забезпечують безперервність обміну інформацією навіть у повністю віддалених командах.

Agile також передбачає регулярні зустрічі синхронізації з більш широкими групами - наприклад, спільні огляди інкременту, Product Backlog Refinement або робота в гільдіях у Spotify-моделі, коли спеціалісти з різних команд обмінюються досвідом.

3.4. Інструменти та практики співпраці в Agile-команді

Ефективна співпраця в Agile-командах значною мірою залежить від правильно підібраних інструментів, які забезпечують прозорість, швидку комунікацію та безперервний потік роботи. У сучасних проєктах використовуються як фізичні, так і цифрові інструменти, проте в умовах віддаленої роботи цифрові рішення стали основою командної взаємодії.

Одними з головних інструментів є таск-трекери: Jira, Trello, Azure DevOps, ClickUp (рис.3.4). Вони дозволяють створювати беклог, вести дошки задач, стежити за статусом роботи та забезпечують повну прозорість процесу. Також вони підтримують Scrum та Kanban-підходи, що робить їх універсальними для різних команд [21, 27].

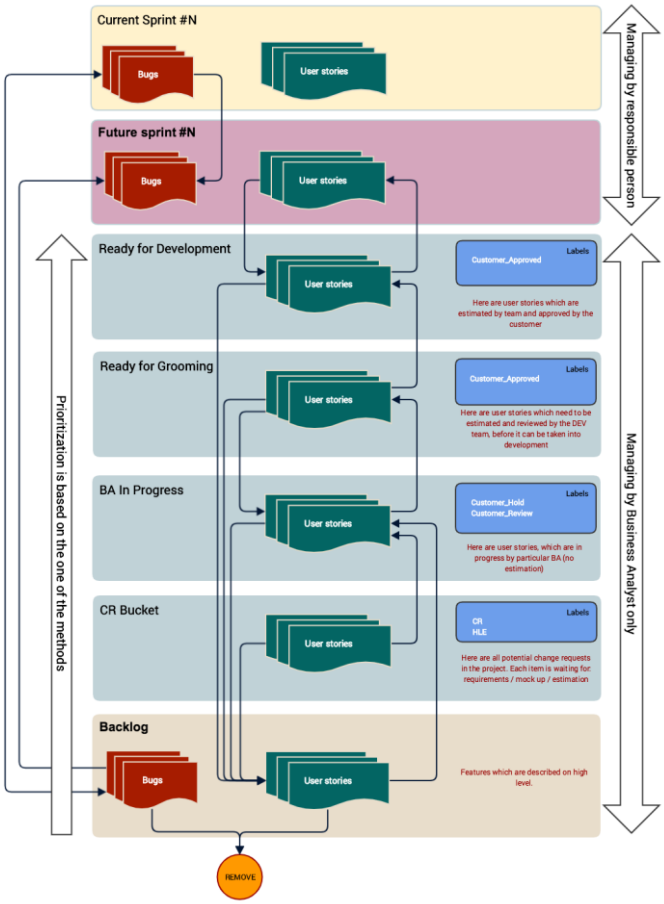


Рис. 3.4 – Таск-трекер Jira як приклад інструменту для управління беклогом та задачами

Для комунікації використовуються чати та відеосервіси: Slack, Microsoft Teams, Google Meet, Zoom. Вони підтримують щоденну взаємодію, миттєвий обмін інформацією та проведення синхронізацій. Команди також активно застосовують тематичні канали, голосові кімнати та інтеграції з таск-трекерами.

Важливою практикою є спільна робота з документами - Google Workspace, Confluence, Notion. Ці сервіси дають змогу створювати єдине

джерело правди (single source of truth), де зберігаються специфікації, вимоги, дизайн-рішення та результати ретроспектив [28].

Технічні команди активно використовують інструменти для контролю версій (GitHub, GitLab, Bitbucket) та CI/CD-платформи, які автоматизують збірку й тестування продукту, забезпечують узгодженість роботи та зменшують кількість ручних операцій [28].

Усі ці інструменти працюють найбільш ефективно лише тоді, коли команда застосовує їх у поєднанні з правильними практиками - прозорою комунікацією, регулярними синхронізаціями та спільною відповідальністю за результат.

3.5. Формування культури співпраці в Agile-команді

Культура співпраці є основою успішної Agile-команди і формує спосіб взаємодії, прийняття рішень та ставлення до роботи. На відміну від традиційних моделей, де акцент робиться на контролі та ієрархії, у Agile головну роль відіграють довіра, прозорість та взаємоповага. Саме культура визначає, наскільки швидко команда зможе адаптуватися, навчатись і долати складні ситуації (рис.3.9).

Ключовим елементом є відкрита комунікація - здатність учасників вільно висловлювати думки, обговорювати проблеми та пропонувати рішення без страху критики. Команди з високим рівнем відкритості швидше виявляють ризики, менше допускають помилок та активно вдосконалюють процеси.

Рис. 3.9 – Компоненти культури співпраці в Agile-команді

Важливою складовою є взаємна підтримка. У членів Agile-команди має бути розуміння, що успіх інкременту - це спільна відповідальність, а не змагання між окремими виконавцями. Така атмосфера позитивно впливає на мотивацію та залученість.

Не менш значущим аспектом є культура навчання та експериментів. Agile заохочує постійне покращення, перевірку нових ідей і швидке випробування гіпотез. Це знижує страх помилок та сприяє інноваційності.

Для підтримання культури співпраці Agile-команди активно використовують ретроспективи - зустрічі, де аналізують сильні та слабкі сторони роботи, формують конкретні кроки для покращення й закріплюють позитивні практики [28,29].

3.6. Методи оцінювання ефективності Agile-команд

Оцінювання ефективності Agile-команд суттєво відрізняється від традиційних підходів, де основний акцент робився на індивідуальній продуктивності. В Agile ключовим є результат команди, якість інкременту та здатність стабільно постачати цінність користувачам. Основні метрики спрямовані на аналіз потоку роботи, швидкості поставки, стабільності процесу та задоволеності стейкхолдерів.

Одним із найпоширеніших показників є Velocity - кількість story points, які команда завершує за спринт. Velocity використовується для планування, але не для порівняння команд. Важливо, щоб швидкість була стабільною і передбачуваною [30].

Другим важливим показником є Cycle Time - час, який витрачається на виконання задачі від моменту початку роботи до її завершення. Чим коротший і стабільніший цикл, тим ефективніший процес. У Kanban Cycle Time є ключовою метрикою оптимізації потоку [30] .

Також активно використовується Burndown Chart, який показує, як команда рухається до завершення спринту. Він дозволяє швидко помітити ризики зриву дедлайнів та потребу в корекції планів.

У масштабованих Agile-середовищах застосовують Lead Time, який вимірює час між появою вимоги й її поставкою до користувача. Це показує загальну ефективність усієї системи, а не окремих команд.

Окрему увагу приділяють якісним показникам: кількість дефектів, рівень технічного боргу, відповідність Definition of Done, відгуки користувачів. Саме вони впливають на довгострокову ефективність роботи.

Не менш важливою є оцінка командної взаємодії - через ретроспективи, внутрішні опитування, аналіз комунікації та ступінь задоволеності учасників процесом.

3.7 Практичне застосування підходів до формування та управління Agile-командою

У межах виконання магістерської роботи розглянуті теоретичні підходи до формування та управління Agile-командою були практично апробовані у власному проєкті з розробки програмного забезпечення, який реалізовувався з використанням Agile-підходів.

На початковому етапі проєкту спостерігалися проблеми, пов’язані з нечітким розподілом ролей, нерегулярною комунікацією між учасниками команди та складністю відстеження виконання завдань. Це негативно впливало на терміни виконання робіт і прозорість процесу розробки.

У процесі роботи було впроваджено елементи Scrum, зокрема визначено ключові ролі (Product Owner, Scrum Master), організовано роботу ітераціями та застосовано регулярні командні зустрічі. Для управління завданнями використовувався таск-трекер, що дозволило структурувати беклог і підвищити контроль за виконанням задач [30,31].

Крім того, під час формування команди було враховано T-shaped модель компетенцій, що сприяло більш гнучкому розподілу обов’язків та взаємозамінності учасників команди.

У результаті впровадження зазначених підходів покращилась координація роботи команди, зросла прозорість процесів та зменшилась кількість організаційних затримок. Команда отримала можливість швидше реагувати на зміни вимог та стабільно досягати запланованих результатів, що підтверджує доцільність застосування Agile-підходів у проєктах з розробки програмного забезпечення.

Висновки

У третьому розділі було розглянуто ключові аспекти формування та роботи Agile-команд, що є фундаментом успішної розробки програмного забезпечення у гнучких підходах. Аналіз показав, що ефективна команда - це не просто група фахівців, а цілісна система з узгодженими ролями, прозорою комунікацією та спільною відповідальністю за результат.

Визначено, що формування команди починається з підбору фахівців із взаємодоповнювальними компетенціями, здатних до самоорганізації та роботи у кросфункціональному середовищі. Принципи психологічної безпеки, відкритості та підтримки є необхідною умовою для розвитку команди та підвищення її ефективності.

Ключові ролі - Product Owner, Scrum Master і команда розробки - забезпечують цілісність процесу: від визначення цінності до створення якісного інкременту продукту. Їх компетенції формують баланс між бізнес-очікуваннями та технічною реалізацією.

Особливу увагу було приділено моделям взаємодії та інструментам співпраці, які дозволяють зберігати прозорість, прогнозованість та узгодженість роботи. Цифрові сервіси, task-трекери й регулярні синхронізації формують основу якісної комунікації як у локальних, так і у віддалених командах.

Розглянуті методи оцінювання ефективності показали, що Agile-команди оцінюються не за індивідуальними показниками, а за здатністю колективно створювати стабільну та передбачувану цінність. Метрики Velocity, Cycle Time, Lead Time та якісні показники є інструментами постійного вдосконалення.

Таким чином, розділ 3 продемонстрував, що успішне формування Agile-команди ґрунтується на поєднанні компетентного складу, зрілої комунікації, сучасних інструментів та культури співпраці. Ці елементи разом створюють основу для ефективної реалізації Agile-проєктів у динамічному середовищі IT.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ПІД ЧАС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення в сучасних умовах пов’язана з інтенсивною інтелектуальною працею, тривалим перебуванням за комп’ютерною технікою та високим рівнем інформаційного навантаження. Незважаючи на відсутність прямої фізичної небезпеки, діяльність програміста супроводжується низкою шкідливих і небезпечних факторів, що можуть негативно впливати на стан здоров’я працівників. Саме тому питання охорони праці в сфері інформаційних технологій є актуальними та потребують детального розгляду.

Охорона праці в ІТ-діяльності спрямована на створення безпечних і комфортних умов праці, зниження рівня професійних ризиків, запобігання перевтомі, професійним захворюванням та підвищення продуктивності праці. В умовах Agile-проєктів, де команда розробки працює в режимі постійної комунікації, ітерацій та дедлайнів, значення правильно організованого робочого середовища суттєво зростає.

4.1. Характеристика робочого місця програміста

Робоче місце програміста являє собою комплекс технічних, ергономічних та організаційних елементів, які забезпечують виконання професійних обов’язків. Основним інструментом праці є персональний комп’ютер або ноутбук, а також допоміжні засоби — монітор, клавіатура, миша, гарнітура та програмне забезпечення.

Праця програміста має переважно сидячий характер і відноситься до розумової діяльності з високою концентрацією уваги. Протягом робочого дня спеціаліст здійснює аналіз вимог, написання коду, тестування, участь у командних зустрічах та комунікацію з іншими членами Agile-команди. Тривале перебування в статичній позі та інтенсивна робота з візуальною інформацією

можуть призводити до швидкої втоми, зниження працездатності та погіршення самопочуття.

Особливістю сучасних Agile-команд є використання як офісного формату роботи, так і віддаленого або гібридного режиму. У таких умовах відповідальність за організацію безпечного робочого місця частково покладається не лише на роботодавця, а й на самого працівника [32].

4.2. Небезпечні та шкідливі фактори під час роботи програміста

Професійна діяльність програміста не пов’язана з безпосередніми виробничими ризиками, однак характеризується наявністю низки шкідливих факторів, які при тривалому впливі можуть негативно позначатися на фізичному та психоемоційному стані працівника. Основними небезпечними та шкідливими факторами в роботі розробника програмного забезпечення є зорове перенапруження, статичне навантаження опорно-рухового апарату, психоемоційна напруга, а також вплив електромагнітного випромінювання від комп’ютерної техніки.

Зорове навантаження виникає внаслідок тривалої концентрації уваги на екрані монітора, роботи з дрібними елементами інтерфейсу та текстовою інформацією. Недостатнє або надмірне освітлення, відблиски на екрані, неправильні налаштування яскравості й контрастності монітора сприяють швидкій втомі очей, зниженню гостроти зору та появі головного болю.

Статичне фізичне навантаження пов’язане з тривалим перебуванням у сидячому положенні. Відсутність рухової активності протягом робочого дня може призводити до порушень постави, болю в спині та шиї, розвитку захворювань опорно-рухового апарату. Особливо негативно на стан здоров’я впливає неправильно підібране робоче крісло або робочий стіл.

Психоемоційна напруга є характерною особливістю роботи в Agile-проєктах, де команда функціонує в умовах обмежених термінів, частих змін вимог та високої відповідальності за результат. Постійні дедлайни, необхідність

швидкого прийняття рішень і командна взаємодія можуть викликати стрес, емоційне вигорання та зниження мотивації.

Окрему увагу слід приділити впливу електромагнітного випромінювання, яке створюється електронними пристроями. Хоча сучасна комп'ютерна техніка відповідає санітарним нормам, тривале перебування поблизу обладнання потребує дотримання регламентованих режимів праці та відпочинку.

4.3. Вимоги до організації робочого місця програміста

Раціональна організація робочого місця програміста є одним із ключових чинників забезпечення безпечних умов праці та збереження працездатності працівника. Вона повинна відповідати санітарно-гігієнічним, ергономічним і технічним вимогам, встановленим нормативними документами.

Робоче місце має бути обладнане зручним робочим столом та ергономічним кріслом з можливістю регулювання висоти сидіння, кута нахилу спинки та підлокітників. Висота робочого столу повинна забезпечувати природне положення рук під час роботи з клавіатурою, що сприяє зменшенню навантаження на м'язи плечового поясу та зап'ясть.

Монітор комп'ютера слід розташовувати на відстані приблизно 50–70 см від очей користувача, при цьому верхній край екрана має знаходитися на рівні очей або трохи нижче. Таке розміщення дозволяє зменшити напруження м'язів шиї та зоровий дискомфорт. Рекомендується використовувати монітори з матовим покриттям і достатньою роздільною здатністю для тривалої роботи.

Освітлення робочого місця повинно бути рівномірним та достатнім, без різких тіней і відблисків на екрані. Перевага надається поєднанню природного та штучного освітлення. Джерела світла необхідно розташовувати таким чином, щоб вони не створювали прямого засліплення та не відбивалися від поверхні монітора.

Мікроклімат у приміщенні має відповідати нормативним показникам температури, вологості та швидкості руху повітря. Оптимальна температура для

офісних приміщень становить 20–24 °С, а відносна вологість повітря — 40–60 %. Дотримання цих параметрів сприяє зменшенню втоми та підвищенню концентрації уваги працівника.

4.4. Заходи з охорони праці під час роботи програміста

Для забезпечення безпечних умов праці програмістів та зменшення негативного впливу шкідливих факторів необхідно впроваджувати комплекс організаційних, технічних і профілактичних заходів. Дані заходи спрямовані на збереження здоров’я працівників, підвищення рівня їх працездатності та запобігання професійним захворюванням.

Одним із ключових заходів є раціональна організація режиму праці та відпочинку. При роботі за комп’ютером рекомендується робити короткі перерви тривалістю 5–10 хвилин кожну годину або після 50–60 хвилин безперервної роботи. Під час перерв доцільно виконувати прості фізичні вправи для зняття напруження м’язів спини, шиї та рук, а також вправи для очей.

Важливе значення має проведення первинного та періодичного інструктажу з охорони праці. Працівники повинні бути ознайомлені з правилами безпечної роботи з комп’ютерною технікою, вимогами до організації робочого місця та основними заходами профілактики професійних ризиків. Дотримання встановлених норм і рекомендацій сприяє зменшенню рівня травматизму та професійних захворювань.

З метою зниження психоемоційного навантаження в Agile-командах доцільно впроваджувати практики здорової командної взаємодії, чітке планування завдань, реалістичні дедлайни та підтримку балансу між роботою й особистим життям. Регулярні ретроспективи дозволяють виявляти проблеми в організації праці та своєчасно вживати коригувальних заходів [33].

Додатковими заходами з охорони праці є забезпечення працівників якісним обладнанням, регулярна перевірка технічного стану комп’ютерної техніки, підтримання належного мікроклімату в приміщеннях і створення умов

для комфортної віддаленої роботи. Комплексний підхід до охорони праці дозволяє мінімізувати ризики та забезпечити стабільну й ефективну діяльність команд розробки програмного забезпечення.

РОЗДІЛ 5 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА В ІТ-ДІЯЛЬНОСТІ

Сучасна діяльність у сфері інформаційних технологій, зокрема розробка програмного забезпечення, не пов’язана з безпосереднім забрудненням довкілля, однак опосередковано впливає на навколишнє середовище через споживання енергетичних ресурсів, використання електронного обладнання та утворення електронних відходів. У зв’язку з цифровізацією суспільства та зростанням кількості ІТ-проєктів питання екологічної відповідальності ІТ-компаній набуває особливої актуальності.

Охорона навколишнього середовища в ІТ-діяльності спрямована на мінімізацію негативного впливу на екосистему шляхом раціонального використання ресурсів, впровадження енергоефективних рішень і дотримання принципів сталого розвитку. Agile-проєкти, завдяки гнучкості та адаптивності, створюють додаткові можливості для впровадження екологічно відповідальних підходів у процесі розробки програмного забезпечення [21,34].

5.1. Вплив ІТ-діяльності на навколишнє середовище

Основний вплив ІТ-діяльності на довкілля пов’язаний із використанням електронних пристроїв та інфраструктури для зберігання й обробки даних. Комп’ютери, сервери, мережеве обладнання та системи охолодження споживають значну кількість електроенергії, що призводить до збільшення викидів парникових газів, особливо у випадках використання енергії з невідновлюваних джерел.

Дата-центри, які забезпечують функціонування хмарних сервісів і програмних продуктів, є одним із найбільш енергоємних елементів ІТ-інфраструктури. Зростання кількості онлайн-сервісів, мобільних застосунків і віддаленої роботи сприяє збільшенню навантаження на такі об’єкти, що, у свою чергу, підвищує екологічні ризики [21,34] .

Крім енергоспоживання, негативний вплив на довкілля має утворення електронних відходів. Застаріле або несправне комп’ютерне обладнання містить небезпечні речовини, які при неправильній утилізації можуть завдати шкоди навколишньому середовищу та здоров’ю людей.

5.2. Основні екологічні ризики IT-діяльності

Екологічні ризики, пов’язані з діяльністю у сфері інформаційних технологій, мають комплексний характер і зумовлені як технічними, так і організаційними чинниками. Незважаючи на нематеріальний характер програмного забезпечення, його розробка та експлуатація потребують значних ресурсів, що впливає на стан навколишнього середовища [34] .

Одним із ключових екологічних ризиків є зростання споживання електроенергії. Робота серверного обладнання, систем зберігання даних і мережевої інфраструктури потребує безперервного енергопостачання. Це призводить до збільшення викидів вуглекислого газу, особливо у регіонах, де переважає виробництво електроенергії з викопних видів палива.

Суттєвим екологічним ризиком є утворення електронних відходів (e-waste). Комп’ютерна техніка, периферійні пристрої та серверне обладнання мають обмежений термін експлуатації. За відсутності належних програм утилізації та переробки електронні відходи накопичуються, створюючи загрозу забруднення ґрунтів і водних ресурсів небезпечними хімічними речовинами.

Також до екологічних ризиків належить теплове навантаження, яке виникає внаслідок роботи дата-центрів. Для забезпечення стабільної роботи обладнання використовуються потужні системи охолодження, що додатково збільшує споживання енергії та впливає на навколишнє середовище.

Окремо слід відзначити непрямий вплив IT-діяльності на довкілля, пов’язаний із зростанням цифрового споживання, збільшенням обсягів даних і розширенням онлайн-сервісів. У сукупності ці фактори формують нові виклики для забезпечення екологічної сталості в цифрову епоху [34].

5.3. Шляхи зменшення негативного впливу ІТ-діяльності на довкілля

Зменшення негативного впливу ІТ-діяльності на навколишнє середовище можливе шляхом впровадження комплексу організаційних, технологічних та управлінських заходів, спрямованих на раціональне використання ресурсів і підтримку принципів сталого розвитку. У сучасних умовах ІТ-компанії дедалі частіше інтегрують екологічні аспекти у свою корпоративну стратегію [21,34,35].

Одним із ефективних напрямів є підвищення енергоефективності програмного забезпечення та інфраструктури. Оптимізація коду, зменшення надлишкових обчислень і використання сучасних алгоритмів дозволяють знизити навантаження на сервери та скоротити споживання електроенергії. Застосування хмарних технологій із використанням енергоефективних дата-центрів також сприяє зменшенню екологічного сліду.

Важливу роль відіграє впровадження віддаленого або гібридного формату роботи, що є характерним для Agile-команд. Скорочення потреби у щоденних поїздках працівників до офісу дозволяє зменшити викиди шкідливих речовин у атмосферу та знизити навантаження на транспортну інфраструктуру.

Раціональне управління електронними відходами є ще одним важливим напрямом екологічної відповідальності. Повторне використання техніки, її модернізація, а також передача застарілого обладнання на спеціалізовану переробку сприяють зменшенню негативного впливу на довкілля. Використання довговічного та якісного обладнання дозволяє знизити темпи утворення електронних відходів.

Додатково варто зазначити роль екологічної свідомості персоналу. Проведення інформаційних заходів, заохочення відповідального ставлення до ресурсів та впровадження екологічних ініціатив у межах компанії сприяють формуванню культури сталого розвитку в ІТ-організаціях.

5.4. Відповідність ІТ-діяльності екологічним стандартам та принципам сталого розвитку

Сучасна ІТ-діяльність дедалі більше орієнтується на дотримання міжнародних екологічних стандартів і принципів сталого розвитку. Для багатьох компаній екологічна відповідальність стає складовою корпоративної культури та важливим фактором конкурентоспроможності на глобальному ринку.

Одним із ключових міжнародних стандартів у сфері охорони навколишнього середовища є стандарт ISO 14001, який регламентує вимоги до системи екологічного менеджменту організацій. Запровадження цього стандарту дозволяє ІТ-компаніям системно підходити до управління екологічними ризиками, контролювати використання ресурсів та мінімізувати негативний вплив на довкілля [21,36,37].

У контексті Agile-проєктів дотримання екологічних стандартів може реалізовуватися через оптимізацію процесів, зменшення надлишкових операцій та впровадження принципів ощадливого виробництва. Гнучкі методології сприяють раціональному використанню ресурсів завдяки ітеративному підходу, постійному аналізу результатів і швидкому реагуванню на зміни.

Важливу роль відіграє також корпоративна соціальна відповідальність, яка включає екологічні ініціативи, використання «зелених» технологій, підтримку екологічних проєктів і формування екологічної свідомості працівників. ІТ-компанії, що дотримуються принципів сталого розвитку, сприяють збереженню навколишнього середовища та формуванню позитивного іміджу в суспільстві.

Таким чином, інтеграція екологічних стандартів і принципів сталого розвитку в ІТ-діяльність дозволяє зменшити негативний вплив цифрових технологій на довкілля та забезпечити збалансований розвиток галузі в довгостроковій перспективі [21,37].

ВИСНОВКИ

У магістерській роботі було досліджено процеси формування та управління командою розробки програмного забезпечення в Agile-проєктах, визначено ключові принципи роботи команд, розглянуто особливості сучасних Agile-методологій і проаналізовано підходи до побудови ефективної взаємодії в умовах високої динаміки IT-середовища.

У першому розділі було здійснено огляд теоретичних засад управління командами в проєктній діяльності та еволюції підходів до організації розробки програмного забезпечення. Показано, що традиційні моделі з жорсткою ієрархією дедалі частіше поступаються місцем гнучким підходам, які забезпечують адаптивність, швидкість прийняття рішень та підвищення якості продукту. Аналіз літератури продемонстрував, що Agile виник як відповідь на необхідність створення продуктів в умовах невизначеності та частих змін, а формування ефективної команди є ключовим фактором успішного застосування гнучких методів.

Другий розділ був присвячений аналізу Agile-методологій, що застосовуються в сучасних IT-проєктах. Особливу увагу приділено Scrum, Kanban, Lean та масштабованим підходам - SAFe, LeSS, Nexus та Spotify-моделі. Порівняльний аналіз показав, що кожен фреймворк має свою унікальну структуру, інструменти та філософію, проте всі вони спрямовані на досягнення стабільного потоку цінності, підвищення продуктивності та покращення взаємодії між учасниками команди. Розглянуті моделі масштабування Agile дозволяють ефективно координувати велику кількість команд, підтримуючи узгодженість інкрементів, прозорість процесів і стратегічну цілісність продукту.

У третьому розділі було досліджено принципи формування високоефективних Agile-команд, охарактеризовано ключові ролі - Product Owner, Scrum Master та Developers - та їхні компетенції. Визначено, що фундаментом успішної роботи команди є кросфункціональність, самоорганізація, психологічна безпека, спільна відповідальність і прозора

комунікація. Окрему увагу приділено моделям взаємодії та цифровим інструментам, які забезпечують узгодженість роботи як у локальних, так і у розподілених командах. Також проаналізовано методи оцінювання ефективності Agile-команд, які базуються не на індивідуальних показниках, а на здатності команди створювати стабільну цінність - Velocity, Cycle Time, Lead Time, якісні метрики та інструменти ретроспектив.

Узагальнюючи результати дослідження, можна зробити висновок, що успішне формування та управління командою в Agile-проєктах ґрунтується на поєднанні технічної компетентності, зрілої командної культури та ефективної комунікації. Agile-команди, які працюють у середовищі довіри, відкритості та взаємної підтримки, здатні швидше адаптуватися до змін, приймати якісні рішення та забезпечувати високу результативність. Використання сучасних фреймворків і методів управління, а також регулярний аналіз ефективності сприяє підвищенню продуктивності та забезпечує створення конкурентоспроможного програмного продукту.

Таким чином, проведене дослідження підтверджує, що Agile є не лише технологічним, а й культурним підходом, який формує нову модель командної взаємодії та дозволяє компаніям ефективно працювати в умовах динамічного ринку. Отримані результати можуть бути використані для подальшого вдосконалення командної роботи, оптимізації процесів розробки та впровадження Agile-культури в організаціях різного масштабу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Beck, K. et al. *Manifesto for Agile Software Development*. Agile Alliance, 2001.
2. Schwaber, K., Sutherland, J. *The Scrum Guide*. Scrum.org, 2020.
3. Cohn, M. *Agile Estimating and Planning*. Pearson, 2005.
4. Rubin, K. S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley, 2013.
5. Poppendieck, M., Poppendieck, T. *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 2003.
6. Anderson, D. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.
7. Larman, C., Vodde, B. *Large-Scale Scrum: More with LeSS*. Addison-Wesley, 2016.
8. Scaled Agile Inc. *SAFe 6.0 Framework*.
9. Kniberg, H., Ivarsson, A. *Scaling Agile @ Spotify*. Spotify Engineering Culture, 2012.
10. Cockburn, A. *Agile Software Development: The Cooperative Game*. Addison-Wesley, 2006.
11. Highsmith, J. *Adaptive Software Development*. Dorset House, 2000.
12. Denning, S. *The Age of Agile*. AMACOM, 2018.
13. Shore, J., Warden, S. *The Art of Agile Development*. O'Reilly Media, 2008.
14. Sutherland, J. *Scrum: The Art of Doing Twice the Work in Half the Time*. Currency, 2014.
15. Brown, T. *Change by Design*. HarperBusiness, 2009.
16. VersionOne. *14th Annual State of Agile Report*, 2020.
17. Atlassian. *Agile Coach: Resources & Frameworks*.
18. Team Topologies. *Skelton, M., Pais, M.* IT Revolution Press, 2019.
19. Standish Group. *Chaos Report*, 2020.

20. PMI. *Agile Practice Guide*. Project Management Institute, 2017.
21. Мартинюк, І. Agile-підходи в управлінні ІТ-проєктами. *Вісник КНУТД*, 2021.
22. Коваленко, О. Формування Agile-команд у сучасних компаніях. *Менеджмент і підприємництво*, 2020.
23. Гнатюк, О. Гнучкі методології в розробці ПЗ. *Економіка та суспільство*, 2021.
24. Петренко, Д. Особливості застосування Scrum у продуктових командах. *IT-менеджмент*, 2022.
25. Сисоєва, С. Технології командної роботи в цифрову епоху. *Педагогіка і психологія професійної освіти*, 2020.
26. Ukrainian IT Association. *Agile Practices in Ukrainian Tech Companies*, 2023.
27. Jira Official Documentation. Atlassian, 2024.
28. Google. *Project Aristotle: The Five Keys to a Successful Team*, 2018.
29. DevOps Research & Assessment (DORA). *Accelerate State of DevOps Report*, 2021.
30. Harvard Business Review. *How to Build a Great Agile Team*, 2019.
31. McKinsey & Company. *Agile Transformation in Large Enterprises*, 2020.
32. Deloitte. *Global Technology Trends Report*, 2022.
33. BCG. *Agile at Scale: Organizational Adaptation Strategies*, 2021.
34. OECD. *Digital Transformation and Agile Management*, 2020.
35. ISO/IEC 26555:2023. *Agile Methods for Software and Systems*.
36. Про охорону навколишнього природного середовища : Закон України від 25.06.1991 № 1264-XII. URL: https://zakon.rada.gov.ua/laws/show/1264-12#Text
37. Стандарт ISO 14001. https://www.dqsglobal.com/uk/doslidzhujte/blog/sfera-zastosuvannya-v-iso-14001-scho-vimagae-standart