МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІННОВАЦІЙНИХ ОСВІТНІХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А. С. Савченко
«_____» _____ 2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ
«МАГІСТРА»
ЗА СПЕЦІАЛІЗАЦІЄЮ «ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)»

Тема: "Методи координації команд при розробці великих програмних проектів в гнучких технологіях"

Виконавиця: студентка групи УС-201Мз Бабич Яна Олексіївна
(студент, група, прізвище, ім'я, по батькові)
Керівник: к.т.н., доцент Харченко Олександр Григорович
(науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: _____ Райчев І. Е.
(підпис) (П.І.Б.)

КИЇВ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут Інноваційних освітніх технологій

Кафедра Комп'ютерних інформаційних технологій

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп'ютерні науки»

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____(А.С. Савченко)

"_____" _____ 2020 р.

ЗАВДАННЯ

на виконання дипломної роботи студентки

Бабич Яни Олексіївни

(П.І.Б. випускника)

1. **Тема роботи:** «Методи координації команд при розробці великих програмних проектів в гнучких технологіях» затверджена наказом ректора від 06.10.2020р. за № 1939/ст.
2. **Термін виконання роботи:** з 05.10.2019 по 31.12.2019
3. **Вихідні дані роботи:** визначення категорій основних координаційних складнощів, моделі процесу ефективної координації, адаптація моделі процесів управління координації до використання у гнучких технологіях розробки програмного забезпечення за умови використання їх для великомасштабних проектів.
4. **Зміст пояснювальної записки:** методології розробки як різність поглядів на життєвий цикл розробки програмного продукту. Процес поєднання гнучких і планових технологій в умовах глобальної розробки. Процеси відбору та реінжинірингу архітектури в ітераціях гнучких методів як важілі координації. Методи координації команд та управління знаннями при розробці великих проектів в гнучких технологіях.

5. **Перелік обов'язкового ілюстративного матеріалу:** таблиці, рисунки, діаграми, графіки, що відображають життєві цикли та структури процесів різних підходів до розробки програмних продуктів, процес трансформації та резолюцій при впроваджені гнучких технологій, процеси в адаптованому Scrum, алгоритми оцінки та рефакторингу архітектури, модель процесу управління координацією.
6. **Календарний план**

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Виконання детального аналізу літератури та Інтернет-джерел згідно із темою роботи.	05.10.2020 – 10.10.2020	
2	Розробка та затвердження плану дипломної роботи.	11.10.2020 – 15.10.2020	
3	Огляд методологій розробки життєвого циклу програмних продуктів та їхнього використання.	16.10.2020 – 18.10.2020	
4	Задача процесу поєднання гнучких і планових технологій у великомасштабних програмних продуктах.	19.10.2020 – 29.10.2020	
5	Огляд методів координації команд та розробка моделі процесу координації в умовах гнучких технологій.	30.10.2020 – 10.11.2020	
6	Розробка звітності.	11.11.2020 – 30.11.2019	
7	Технічне оформлення пояснювальної записки.	01.12.2020 – 10.11.2019	
8	Підготовка до захисту дипломної роботи.	10.12.2020 – 20.12.2020	

7. **Дата видачі завдання:** «05» жовтня 2020 р.

Керівник дипломної роботи: _____ Харченко О.Г.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання: _____ Бабич Я.О.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Методи координації команд при розробці великих програмних проектів в гнучких технологіях»: 105 сторінок, 27 рисунків, 14 таблиць, 40 літературних джерел.

Об'єкт дослідження: координація команд у межах великомасштабних гнучких технологій розробки програмних продуктів.

Предмет дослідження: підходи для підвищення узгодженості між командами в умовах великомасштабної гнучкої розробки програмного забезпечення.

Мета роботи: огляд використання гнучких та планових технологій у межах глобальної розробки, формалізація процесів прийняття рішень про розробку, забезпечення високого рівня координації та якості програмних продуктів шляхом розробки математичної моделі.

Методи дослідження: обробка літературних джерел, порівняльний аналіз досвіду великих компаній.

Результати магістерської роботи рекомендується використовувати під час проведення наукових досліджень та в практичній діяльності фахівців в межах глобальної розробки із залученням гібридних технологій.

Ключові слова: ГНУЧКІ ТЕХНОЛОГІЇ, ТРАДИЦІЙНІ МЕТОДОЛОГІЇ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РЕФАКТОРИНГ, ПАТЕРНИ АРХІТЕКТУРИ, ВЕЛИКОМАСШТАБНІ ПРОЕКТИ, ГЛОБАЛЬНА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ШИРОКОМАСШТАБНА ГНУЧКА ТРАНСФОРМАЦІЯ, УПРАВЛІННЯ КОМАНДАМИ, КООРДИНАЦІЯ МІЖ КОМАНДАМИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ	Програмне забезпечення
ПП	Програмний продукт
БКС	Багатоканальна система
ЦК	Центральна команда
КР	Командний рівень
МКР	Міжкомандний рівень
СФ	Слабо формалізовані (системи, критерії)

ВСТУП

Актуальність обраної теми полягає в тому, що на сьогоднішній день невеликі проекти та команди, які розміщені разом, не є обмеженням для залучення гнучких технологій. Внаслідок цього проекти розробки програмного забезпечення (ПЗ) зазнали змін щодо технічної та організаційної складності, включаючи велику кількість зацікавлених сторін, учасників програм, вимог та складних взаємозалежностей між завданнями. І саме здатність керувати цією складність та координація є ключем до успіху великих програм, однак розуміння в цьому напрямку у рамках глобальної розробки обмежене на даний час. Тому досягнення ефективної координації є однією з найактуальніших проблем у масштабній розробці ПЗ.

На основі цього була поставлена мета дипломної роботи, що полягає у вивчені технологій глобальної розробки програмних продуктів (ПП) й пов’язаних із цим складнощів координації команд, аби на основі цього розробити математичну модель процесу управління.

Основним завданням є формалізація процесів прийняття рішень про розробку, забезпечення високого рівня координації та якості ПП шляхом розробки моделі процесу управління координації. Об’єктом даної дипломної роботи є координація команд у межах великомасштабних гнучких технологій розробки.

Предметом дослідження є гнучкі технології розробки ПЗ у межах великомасштабних проектів та проблеми, пов’язані з процесами адаптації традиційних підходів розробки, підвищення якості ПП, а також підвищення оптимізації для процесів прийняття рішень та узгодженості між командами в умовах глобальної розробки.

Досягти поставленої цілі дипломного проектування допомогла обробка літературних джерел та порівняльний аналіз досвіду використання гнучких технологій великомасштабними ІТ-компаніями. Для підвищення ефективності процесів прийняття рішень про розробку, забезпечення високого рівня координації

та якості ПП необхідно формалізувати ці процеси шляхом розробки математичної моделі процесу управління координацією.

Дипломна робота складається з чотирьох розділів.

У першому розділі розглянуті теоретичні відомості про методології розробки ПЗ, а саме - технології гнучкого та традиційного підходів.

Другий розділ описує особливості поєднання та співіснування гнучких і планових технологій в умовах великомасштабної розробки, поняття глобальної розробки, складнощі, які виникають у ході широкомасштабної трансформації компаній та можливі резолюції, а також погляд на гібридну розробку як спосіб впровадження обох підходів до розробки.

У третьому розділі відображено процеси відбору та реінжинірингу архітектури в ітераціях гнучких методів, розглянуті поняття гнучкої архітектури та адаптованого Scrum. Розкрито передумови виникнення потреб у координації з причини виникнення змін в архітектурі завдань команди, тобто надано огляд процесу рефакторингу та оцінку вибору використовуваних патернів.

У четвертому розділі розглянуто особливості забезпечення цілісності та актуальності інформації при розробці великомасштабних проектів, використовуючи існуючі координаційні процеси в межах командної роботи та методи полегшеного впровадження гнучких технологій у великомасштабні організації розробки ПЗ, а також можливі виникаючі складнощі на основі реального досвіду світових компаній.

У розділі наведена адаптація моделі процесу управління координацією в рамках глобальної розробки із залученням гнучких технологій як організацію управління у системах із умовами, що слабко формалізуються.

РОЗДІЛ 1

МЕТОДОЛОГІЇ РОЗРОБКИ ЯК РІЗНІСТЬ ПОГЛЯДІВ НА ЖИТТЄВИЙ ЦИКЛ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

Вже протягом кількох десятиліть задача визначення підходів та способів підвищення продуктивності, якості та надійності розробки ПП є актуальною. При цьому, кожен фахівець має свої власні погляди щодо покращення процесу, який є важко передбачуваним. В той час, як одні намагаються вирішити питання шляхом систематизації та формалізації, другі – застосувати методи управління проектами та методи програмної інженерії, а треті – дотримуються думки, що розробка продукту повинна підлягати постійному контролю замовника [1]. До цього ж, методологія розробки ПЗ повинна бути порівняна із складністю структури самого ПП. Інакше, якщо обрана методологія є невиправдано складною для даного продукту, то це лише збільшить вартість розробки.

Життєвий цикл ПЗ представляє собою стадії, через які проходить ПП від появи ідеї до його подальшої підтримки та «смерті». Тобто, у ПП, подібно до живої істоти, є свій життєвий цикл, який багато в чому і зумовлює методології розробки ПЗ [1].

Принципи, поняття, ідеї, методи, способи та засоби – вся ця сукупність визначає методологію розробки й у підсумку визначатиме стиль розробки ПЗ. Іншими словами, методологія є способом реалізації певного стандарту, на вибір якої можуть впливати розмір команди, складність певного проекту, зрілість і стабільність процесів в компанії. Таким чином, методології - це ядро теорії керування розробкою якого-небудь ПЗ [2].

1.1. Огляд Agile технологій

На сьогоднішній день інформаційні технології проникли і поринули в усі сфери нашого життя, оскільки в світі спостерігається зосередження на цифровізації, й разом із цим приходить усвідомлення важливості ПП. Необхідність підвищення швидкості реагування та покращення можливостей для обслуговування все більшої кількості клієнтських потреб – тиск, який дедалі більше відчувають на собі компанії, що займаються розробкою ПЗ. І це призводить не лише до необхідності привнесення деякої гнучкості бізнесу з метою впровадження нової продукції на ринок, але й до необхідності еволюціонувати та модифікувати вже існуючі рішення та послуги [3].

Нові технології, пропозиції та можливості конкурентних фірм, прийняті нові закони, та й загальні коливання ринку, роблять актуальним існування арен для конкурентної боротьби, пов’язаної із швидким постачанням ПЗ підприємством. Переможцем із цього поєдинку може вийти лише той, хто не боїться змін і готовий оновити діяльності, спрямовані на керування розробкою, зокрема, безліч формальних і неформальних процедур, практик, процесів і правил [2]. Ці механізми управління - важлива функція в керуванні та контролі над тим, як ПЗ постачається у виробництво.

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

Віднайти цей баланс все важче із плином часу, тому що відбувається нагромадження тиску з отримання все більшої і більшої швидкості реагування. Виникаючі виклики спрямовують організації до швидкого впровадження гнучких методів у політику організації [2]. Дослідження свідчать про те, що гнучкі практики - домінуючі підходи у багатьох сучасних організаціях розробки ПЗ.

Гнучкість відкриває нові можливості в галузі забезпечення якості ПЗ, оскільки це не лише задоволення потреб клієнтів, але й дає можливість змінювати вимоги аж до рівня розгортання продукту [1].

Гнучкі технології розробки ПЗ обіцяють підтримувати безперервний зворотний зв'язок і враховувати зміни у вимогах протягом всього життєвого циклу розробки ПЗ, підтримувати тісну співпрацю між клієнтами і розробниками, а також забезпечувати ранні і часті поставки програмних функцій, необхідних для системи [2] (рис. 1.1).

Рис. 1.1. Життєвий цикл гнучкої розробки ПЗ

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

Відповідно до маніфесту Agile [3]:

Ми постійно відкриваємо для себе більш досконалі методи розробки ПЗ, займаючись розробкою безпосередньо і допомагаючи в цьому іншим. Завдяки виконаній роботі ми змогли усвідомити, що [3]*:*

Люди і взаємодія *важливіше процесів та інструментів*

Працюючий продукт *важливіше вичерпної документації*

Співпраця з замовником *важливіше узгодження умов контракту*

Готовність до змін *важливіше проходження попереднім планом*

Тобто, не заперечуючи важливості того, що справа, ми все-таки більше цінуємо те, що зліва [3].

Гнучкі методології можна визначити як групу процесів розробки ПЗ, які є [3] (рис. 1.2):

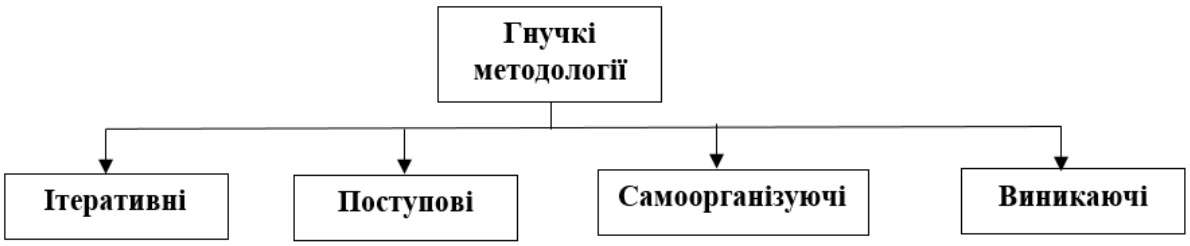

Рис. 1.2. Визначення гнучких методологій

– ***ітераційними*** як спроба вирішення проблем шляхом пошуку послідовних наближень до рішення (наприклад, розроблена повна система змінює функціональність підсистем з кожним випуском внаслідок оновлення вимог);

– ***поступовими*** як підхід, який передбачає розподіл системи на функціональні підсистеми та додавання нової функціональності до загальної системи з кожним випуском;

– ***самоорганізуючими*** як концепція, що дає команді можливість організовувати самостійно процес для найкращого результату завдання (наприклад, взаємодія в команді, динаміка роботи, робочий час команда самостійно розподіляє для найліпшого вирішення);

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

Деякі з відомих методів гнучких технологій розробки ПП - екстремальне програмування (XP), Scrum, кристальні методології Crystal Clear, Kanban, розробка керована функціями, динамічний метод розробки [4]. У цьому розділі буде розглянуто три відомі методи – Scrum, XP, Kanban.

1.1.1. Scrum

Scrum – один із провідних методів гнучких технологій з розробки ПЗ, який є ітеративним та інкрементним підходом до керування проектами. Назва методу – це однойменний термін в грі регбі, де він означає «повернення м'яча поза грою» через командні зусилля [5]. В проектах із Scrum-ом є основними наступні ролі.

Перша роль - це *Scrum-майстер*. Він є тренером, який допомагає команді правильно використовувати процес Scrum для виконання процесів на найвищому рівні. Від традиційного менеджера проекту його відрізняє те, що ця роль не забезпечує щоденне керівництво командою і не призначає завдання окремим особам.

Власник продукту представляє бізнес, клієнтів або користувачів, і направляє команду до правильної мети, аби створити якісний продукт [4]. Робиться це шляхом створення бачення продукту із подальшою передачею цього бачення команді через відставання продукту.

Третя і остання роль - *команда Scrum*. Оскільки людина, яка б вирішувала хто і яке завдання буде виконувати відсутня, то дана команда є такою, що саморганізовується. Ці питання по розподілу задач і зайнятості вирішуються командою в цілому. Scrum-команда є функціональною, що означає, що кожен повинен взяти участь в розробці від ідеї до реалізації [4].

Для того, аби уявити процес взаємозв’язку приведених ролей, можна собі представити гоночний автомобіль [5]. В даному прикладі, нехай команда буде являти собою сам автомобіль, який готовий до руху у будь-якому із напрямків. Водієм є власник продукту, в обов’язки якого входить переконатися, що автомобіль завжди йде в правильному напрямку. Головним механіком в даному процесі є Scrum-майстер, який зберігає автомобіль добре налагодженим.

Якщо говорити про артефакти, то кожному проекту Scrum властиві наступні п’ять [5].

Продукт являє собою первинний артефакт, який приводиться командою до потенційно доступного стану в кінці кожного спринту.

Відставання продукту - це повний перелік функціональних можливостей, які потрібно заімплементуваим до продукту. Команда завжди працює над найціннішими з них згідно із пріоритетам, наданим власником.

Історія користувача – короткий опис функціональності з точки зору користувача або клієнта.

Спринтове відставання – артефакт, який являє собою список завдань для виконання командою, щоб забезпечити функціональність, призначену на певний спринт.

Як додатковий артефакт виступають *діаграми спринтового розгортання* і *діаграми випуску релізів* (діаграмами вигоряння). Завдяки них можна відобразити обсяг роботи, що залишився. Діаграми є ефективним інструментом для визначення завершеності спринту або релізу до бажаної дати.

Scrum акцентує увагу на тому, що "визначені та повторювані процеси працюють лише для вирішення визначених та повторюваних проблем із визначеними та повторюваними людьми у визначених та повторюваних середовищах", а це, як очевидно, неможливо [4]. Саме тому, для вирішення наведеної проблеми процесів, проект розділяється на, так званні, ітерації (спринти).

Тобто, за даної моделі, проекти розробляються «покроково» через визначену серію спринтів (тривалістю 3-4 тижні) [4]. Перед стартом кожного нового спринту збирається зустріч (тривалістю близько 4 годин) задля відбору пріоритезованих історій користувачів, які накопичені у відставанні продукту. Протягом спринту команда із відібраної сукупності бере невеликий набір функцій, щоб реалізувати їх від стадії ідеї до кодованої та перевіреної функціональності.

Кожного дня Scrum-команда проводить коротку зустріч (близько 15 хвилин), щоб кожен член команди міг відповісти на три запитання: «Що я зробив вчора, що я

буду робити сьогодні, і які перешкоди стали на моєму шляху при вирішенні завдання?» [5].

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

Зустріч пріоритезації завдань відставання продукту, щоденна зустріч команди, огляд спринту – контур зворотного зв’язку, який може або призвести до змін у продемонстрованій функціональності продукту, або ж – до перегляду та додавання елементів до відставання продукту [5].

В кінці кожного спринту вся команда приймає участь у діяльності під назвою ретроспектива, щоб поміркувати над закінченим спринтом і визначити аспекти для покращення у роботі.

Scrum-процес ділиться на три частини: фаза аналізу, фаза розробки і фаза огляду (рис. 1.3).

Рис. 1.3. Структура процесу Scrum

Згідно рис.1.3 процес починається з етапу вимог, вони збираються та аналізуються, що дає змогу сформувати задачі до відставання продукту. Відставання продукту пріоритезує найбільш цінні елементи зверху списку при обліку залежностей, елементи з вищим пріоритетом будуть реалізовані в найближчому спринті [4].

Реалізація системи слідує етапу аналізу і займає час в середньому від 2 до 4 тижнів (що і являє собою ітерації). Початок спринту починається з розподілу розробниками задач з верхньої частини відставання продукту і подрібнення їх на завдання, які вони завершують протягом спринту.

Спринти завершуються оглядом результатів ітерації клієнтом. Дані результати оцінюються і визначається чи були досягнуті цілі, встановлені для спринту. Цей етап також супроводжується навчальною сесією, що підкріплюється досвідом, з метою поліпшення практики роботи. На цей час ПЗ має бути протестоване і готове до поставки [5].

1.1.2. Extreme programming

Іншим популярним методом гнучких технологій розробки ПП є *екстремальне програмування (XP)*. Як і інші методи гнучких технологій, він працює завдяки об’єднанню всієї команди за допомогою простих практик із включенням достатнього зворотного зв’язку, а також виступає за коротку ітерацію і часті випуски робочого коду з метою підвищення продуктивності [6]. XP був розроблений для команд, що включають 8-10 учасників, які працюють з об'єктно-орієнтованою мовою програмування.

XP-процес починається зі створення історій користувача замовником, що описують невеликі одиниці функціональності і які можна реалізувати в середньому за тиждень-два процесу тестування та кодування. Замовник, виходячи із кошторису, який йому надають, пріоритезує історії. Реалізація функцій проходить ітераційно, що передбачає поставки замовнику кожні два тижні. Поступово система зростає у функціональності, по частинах, паралельно керуючись замовником на оглядах [4].

Нижче наведено основні практики, що використовуються в XP [6] (рис. 1.4).

Вся команда – практика, яка передбачає, що всі учасники проекту сидять поряд та разом, як члени однієї спільної команди. Сюди входять замовник (представник бізнесу, який надає вимоги та пріоритети), програмісти та тестувальники (кодують, реалізують та визначають приймальні тести), аналітики (допомагають замовнику визначити вимоги). Зазвичай, в команду входить ще і тренер (в обов'язки якого входить допомагати команді дотримуватись вірного шляху) та менеджер (забезпечує ресурси, керує зовнішніми зв'язками, координує діяльність) [6]. Треба зазначити , що жодна з наведених ролей та представників не обов'язково виключно виконує перелічені обов'язки та властивості, кожен у команді XP сприяє будь-яким способом.

Планування гри – практика тісної клієнтської та виконавчої взаємодії, яка дуже рекомендується з метою оцінки та визначення пріоритетів й вимог для наступного випуску. Команда імплементує та поставляє клієнтові виключно узгоджені з замовником історії користувачів [6].

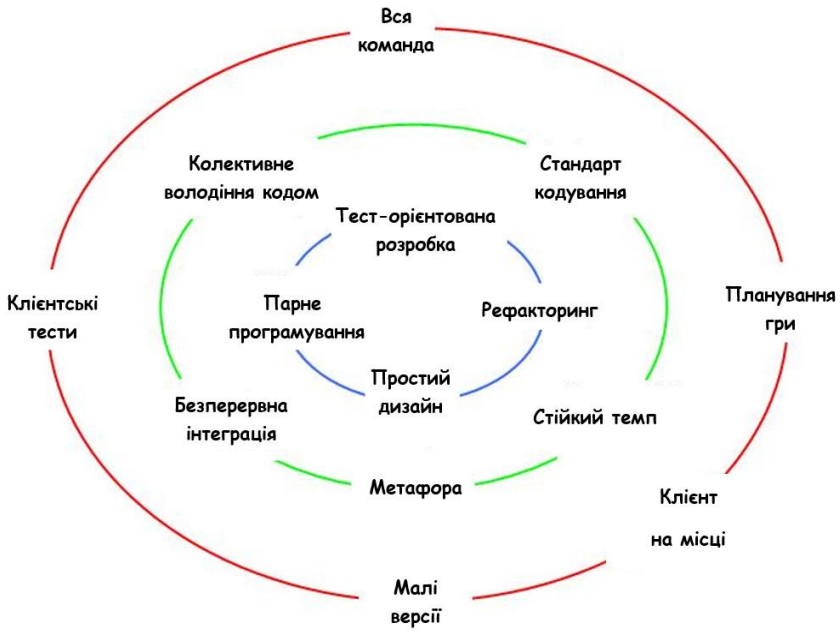

Рис. 1.4. Практики екстремального програмування

Малі версії – практика випуску демо-версії системи після декількох ітерацій з подальшою імплементацією нових функцій в наступних випусках за визначеним терміном (щодня, щотижня тощо).

Клієнтські тести – практика визначення клієнтом одного або кількох автоматизованих приймальних тестів для кожної функції для демонстрації її працездатності [4]. Наведені тести будуються і використовуються командою з метою приведення та перевірки коректності роботи ПЗ.

Простий дизайн – практика заохочення розробників до максимально спрощеного адекватного дизайну системи, який точно підходить для поточної функціональності системи [4].

Парне програмування – практика написання коду двома програмістами, які сидять поряд один з одним на комп'ютері. Це дає змогу та впевненість в тому, що коди переглядаються не однією людиною, що, в свою чергу, призведе до кращого дизайну, процесу тестування та поліпшення коду [6].

Тест-орієнтована розробка – практика, яка передбачає принцип test-first і означає, що розробники пишуть приймальні тести, перш ніж писати сам код, а клієнти, у свою чергу, пишуть функціональні тести для регресії кожної ітерації.

Рефакторинг – практика, яка пропонує на початку проекту розробку простого дизайну із поступовим його покращенням [4]. Подібна практика розглядає розвиток системи шляхом перетворення існуючої, таким чином, щоб всі тести успішно виконалися.

Безперервна інтеграція – практика, за якої система завжди зберігається повністю інтегрованою і весь новий код мерджиться в систему якомога частіше. При цьому, після кожного мерджу проводиться регресія по функціональним тестам.

Колективне володіння кодом – практика, коли всі програмісти володіють спільним кодом і кожен може змінити/покращити якусь його частину [6].

Стандарт кодування – практика, за якої весь код дотриманий згідно загального стандарту кодування і увесь код виглядає з дотриманням одного стилю написання.

Метафора – практика, яка передбачає розробка набору метафор (емоційних характеристик роботи ПЗ) командою для моделювання розроблюваної системи. Використовується загальна система імен, за допомогою якої всі спілкуються і розуміють одне одного.

Стійкий темп – практика, завдяки якій команда сама визначає темпи роботи і може працювати понаднормово, коли вважає це за ефективне з метою максимізації продуктивності із тижня в тиждень [6].

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

1.1.3. Kanban

Kanban - це метод гнучких технологій, який допомагає здійснювати процеси проектування, управління та вдосконалення потокових систем. Його назва походить внаслідок того, що в основі нього лежить використання канбан-механізмів візуальної сигналізації, які дають змогу контролювати готовність для ПЗ. Він дотримується принципу, що робота безперервно протікає через систему, а не організовується у різні часові рамки [7].

Даний метод придатний для використання у ситуаціях, де необхідна організація знань, коли робота є непередбачуваною та коли алгоритм роботи передбачає інкрементну поставку продукту замовнику, як тільки він буде готовий.

Kanban команди дотримуються наступних цінностей [8]:

- ***прозорість*** - відкритий обмін інформацією покращує потік ділової цінності;
- ***баланс*** – збалансування різних аспектів, точок зору та можливостей для досягнення ефективності;
- ***співпраця*** - покращення способу спільної роботи;
- ***орієнтація на клієнта*** – мета оптимізації потоку вартості для споживачів, які є зовнішніми від системи;

- ***потік*** - робота є безперервною у одній часовій рамці;
- ***лідерство*** - здатність надихати інших діяти, необхідне на всіх рівнях;
- ***розуміння*** – пізнання (як індивідуальне, так і організаційне) вихідної точки необхідно для руху вперед та вдосконалення;
- ***угода*** – кожен учасник команди спільно рухається до цілей, поважаючи та враховуючи розбіжності в думках та підходах;
- ***повага*** – цінування та розуміння у команді.

Наступні практики є діяльністю, необхідною для управління системою Kanban (табл. 1.1).

Таблиця 1.1

Практики Kanban

Практика	**Опис**
Візуалізація	Дошка Канбан з метою візуалізації роботи та процесу (рис. 1.5).
Обмеження незавершеного виробництва	Згладження потоку робіт, скорочення термінів виконання, поліпшення якості і частоти доставки шляхом обмеження обсягу виконуваної роботи в системі.
Управління потоком	Максимізація доставки вартості, мінімізація часу виконання і максимальне передбачення – завдання потоку роботи. Ключовий аспект – виявлення та вирішення вузьких місць та блокерів.
Чіткі правила	Обмеження WIP, розподіл потужності, визначення виконаного та інші правила для робочих завдань, що існують на різних етапах процесу.
Цикли зворотного зв'язку	Огляд стратегії, огляд операцій, огляд ризиків, огляд надання послуг, зустріч поповнення, зустріч у Канбані, зустріч з планування доставки.
Спільне поліпшення	Початок з процесу як він є і застосування постійного та

та експериментальний розвиток	поступового вдосконалення.

Дуже пізнаваною практикою Kanban є дошка, яка є обов'язковим елементом даного підходу. Вона завжди знаходиться у вільному доступі, тому кожен член команди в будь-який час бачить на якому етапі перебуває завдання [3].

Kanban дошка підійде і реальна, і віртуальна: можна використовувати просту коркову або програми-трекери на кшталт Trello, Jira, оскільки вона підлаштовується під будь-який процес і застосовується в будь-якій області з метою скласти список справ [7].

Кожен проект має власний план процесів та робіт, який спочатку проходить фазу аналізу і декомпозується на відповідні стовпці-етапи. Наприклад, для процесу створення IT-проекту етапи можуть бути такими, як вказано на рис. 1.5: Історія користувача, Необхідно зробити, В процесі, Тестується, Зроблено. При цьому, імена стовпців можуть змінюватися, не порушуючи методології, важливо лише зберігати їхню послідовність, оскільки поток - це ключова цінність Kanban [7].

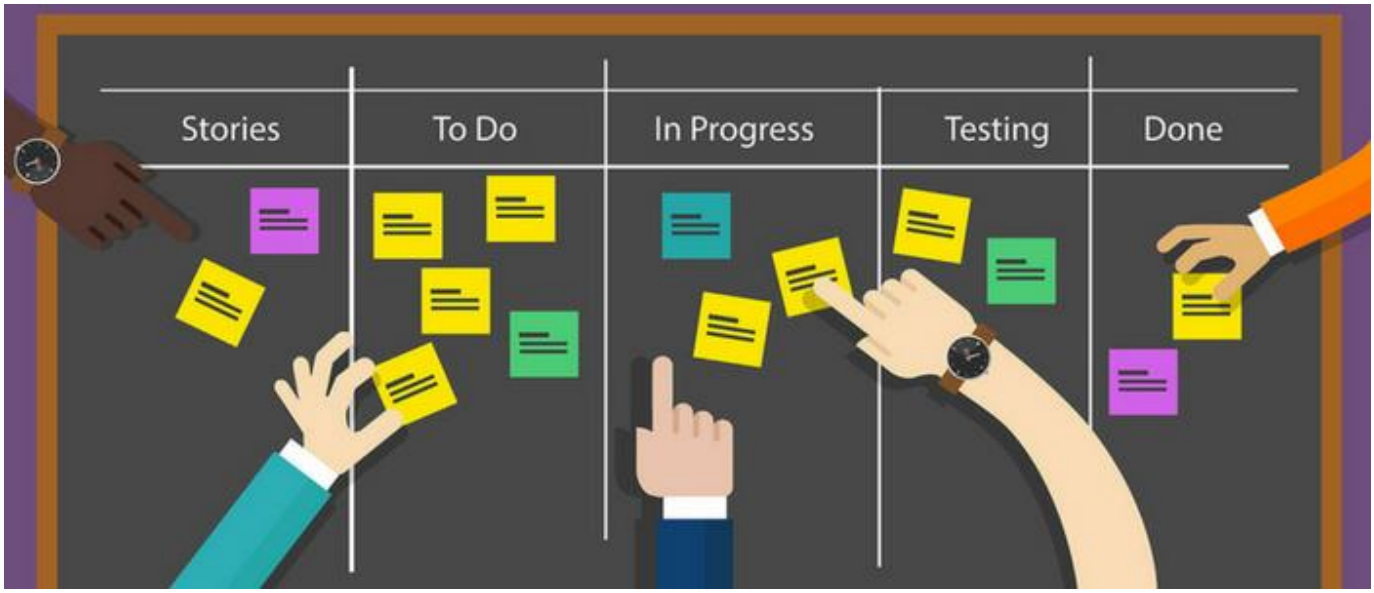

Рис. 1.5. Kanban дошка

Як було зазначено вище, вся робота у Kanban є єдиним потоком і кожна система у процесі роботи відрізняється етапами. Тому найкращий спосіб описати життєвий цикл методу через задіяні петлі зворотного зв'язку [8] (табл. 1.2).

Таблиця 1.2

Цикли зворотного зв’язку Kanban

Каденція зворотного зв’язку	**Частота проведення**	**Мета**
Огляд стратегії	щоквартально	Визначення послуг, що надаються, та контекст, у якому вони є доречними.
Огляд операцій	щомісячно	Визначення балансу між послугами із врахуванням людей та ресурсів з метою максимізації доставки вартості.
Огляд ризиків	щомісячно	Визначення ризиків доставки послуг.
Огляд надання послуг	1 раз на 2 тижні	Визначення та підвищення ефективності послуг (подібно до ретроспективи).
Зустріч поповнення	щотижнево	Визначення предметів роботи команди (подібно до зустрічі планування спринту чи ітерації).
Зустрів у Канбані	щоденно	Координація діяльності команди протягом дня (подібно до дейлі мітингів).
Зустріч з планування доставки	згідно каденції доставки	Відстеження та планування доставки споживачам.

Деякі команди об'єднують ідеали Scrum і Kanban в Scrumban. Зі Scrum беруться спринти фіксованою тривалості і ролі, а з Kanban - концентрацію на тривалості циклу і обмеження кількості одночасно виконуваних завдань [3]. Проте, на етапах адаптації та гнучкої трансформації настійливо рекомендується вибрати ту чи іншу методику і деякий час слідувати виключно їй, адже, час для експериментів знайдеться завжди.

1.2. Огляд традиційних підходів

У той час, коли програмування лише зароджувалося, єдиними учасниками процесу розробки були лише програмісти, які писали програмні функції для полегшення математичних розрахунків або автоматизації інших рутинних дій [8].

На сьогоднішній день все зовсім інакше, сучасні системи досягли таких величезних масштабів і складності, що їхнє створення вимагає долучати до роботи цілі команди фахівців різного профілю: програмісти, аналітики, системні адміністратори, тестувальники і кінцеві користувачі [7]. Через таку велику кількість залучених осіб постає питання ускладненої координації, а для полегшення цього процесу компанії дотримуються певних моделей життєвого циклу ПЗ.

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

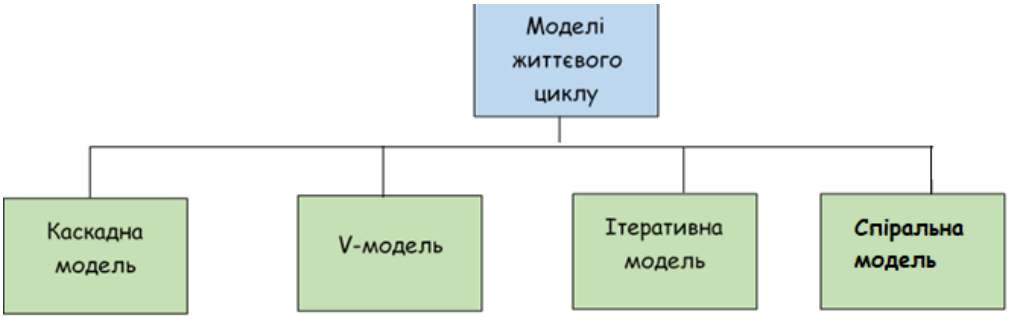

Рис. 1.6. Класифікація моделей життєвого циклу ПП

Може здатися, що індустрія програмної інженерії прийшла, нарешті, до загальної "правильної" моделі. Проте, каскадна модель, багаторазово розкритикована і теорією, і практикою, продовжує зустрічатися в світових компаніях, ітераційна модель є яскравим представником еволюційного погляду, але, в той же час, є єдиною моделю, яка приділяє явну увагу аналізу та попередження ризиків [8]. На сьогоднішній день в різних джерелах наводиться різний список моделей та їхня інтерпретація. Нижче представлені виділені вище чотири моделі - каскадна, V-модель, ітераційна та спіральна.

1.2.1. Каскадна модель

Каскадна (або водоспадна, лінійно-послідовна) модель є найстарішою, найпростішою і найвідомішою моделлю побудови багаторівневого процесу розробки ПП. Вона є дуже легкою і простою для розуміння, але, в той же час, занадто ідеалістичною і є не настільки практичною як раніше [9].

В умовах динамічних та швидко змінюючихся вимог, подібний строго структурований підхід до процесу може з переваги перетворитися в перешкоду на шляху успішного завершення розробки системи [9], тому сьогодні водоспадна модель якщо і застосовується, то це переважно великими компаніями і лише для великих і складних проектів, які передбачають всеосяжний контроль ризиків.

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

Класична модель ділить життєвий цикл на деякий набір фаз, кожну з яких можна розпочати лише після завершення попередньої, тобто вихід однієї фази є входом для наступної [10]. І ось за такої ідеї, процес розвитку є «водоспадом» з послідовним потоком фаз, звідки і походить назва. В цьому і полягає основна відмінність каскадної методології від гнучких конкурентів: Agile, DSDM, Scrum, FDD і т.д.

Різні послідовні фази класичної моделі водоспаду показані нижче (рис.1.7, табл. 1.3).

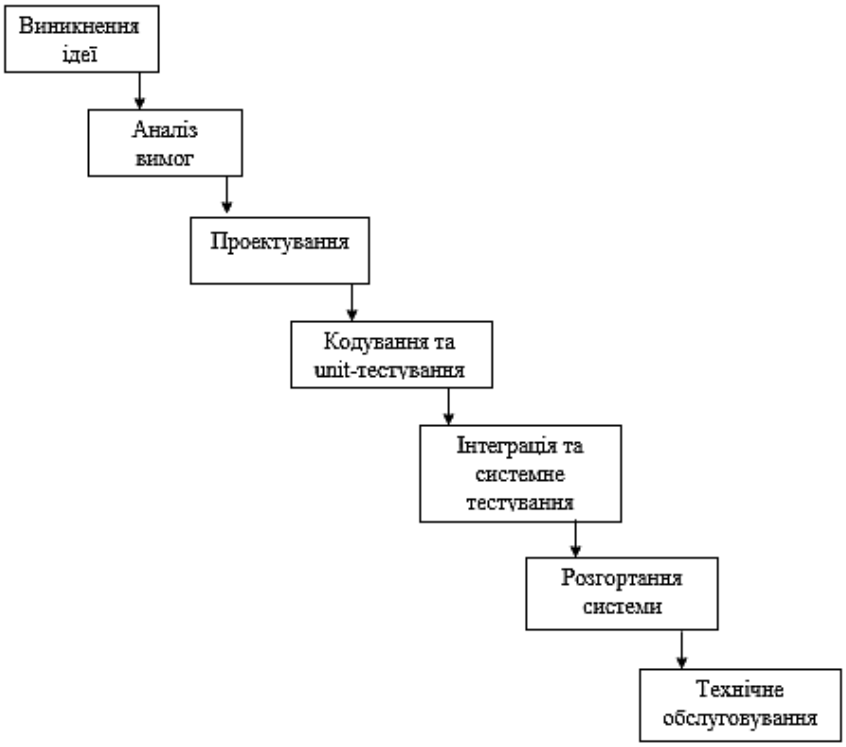

Рис. 1.7. Каскадна модель життєвого циклу ПЗ

Послідовні фази у моделі водоспаду розписано у таблиці 1.3.

Таблиця 1.3

Фази каскадної моделі

Назва фази	**Діяльність фази**
Виникнення ідеї та її обговорення	- розгляд ідеї, що виникла - техніко-економічне обгрунтування - визначення стратегій роботи
Аналіз та специфікація вимог	а) Збір та аналіз вимог: - збір вимог від замовника - аналіз зібраних вимог (усунення неповноти та невідповідностей). б) Специфікація вимог: - задокументовані в документі специфікації вимог до ПЗ;

Продовження таблиці 1.3

Проектування	- перетворення вимог у структуру, яка підходить для реалізації на якійсь мові програмування. - вибір мови програмування (Java, PHP, .net), бази даних (Oracle, MySQL тощо), інших технічні деталей.
Кодування та модульне тестування	- дизайн ПЗ перекладається у вихідний код; - перевірка належної працездатності кожного модуля.
Інтеграція та системне тестування	А) Попередньо заплановані модулі додаються до частково інтегрованої системи та тестується отримана система. Б) Системне тестування: - альфа-тестування - це системне тестування, яке проводить команда розробників. - бета-тестування - це тестування системи, проведене доброзичливим набором клієнтів. - прийомне тестування - замовник проводить приймально-здавальне тестування, щоб визначити чи приймати поставлене ПЗ.
Розгортання системи	- розгортання системи у відповідному середовищі.
Технічне обслуговування	- коригувальне технічне обслуговування: виправлення помилок, які не були виявлені на етапі розробки продукту; - досконале технічне обслуговування: покращення функціональних можливостей системи на основі запиту замовника; - адаптивне обслуговування: перенесення ПЗ для роботи в новому середовищі.

Треба пам’ятати, що кожне розроблене ПЗ відрізняється і вимагає відповідного і не схожого на інші підходу до розробки ПП, який слід застосовувати,

спираючись на внутрішні та зовнішні фактори. Деякі ситуації, коли використання моделі водоспаду є найбільш доцільним [10]:

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

1.2.2. V-модель

V-модель є «поліпшеною» версією розглянутої вище каскадної моделі розробки ПЗ, де виконання процесів відбувається послідовно у, так званій, V-подібній формі. Вона також відома як модель верифікації та валідації [11]:

- *Верифікація* - це статична практика перевірки документів, дизайну, архітектури, коду тощо, яка включає техніку статичного аналізу (огляд) [11], виконану без виконання коду, та відповідає на питання «Чи робимо ми продукт правильно?».
- *Валідація* - це процес оцінки кінцевого продукту, необхідно перевірити, чи відповідає ПЗ очікуванням і вимогам клієнта, яка включає техніку динамічного аналізу (функціональну, нефункціональну) [12] та відповідає на питання «Чи робимо ми продукт правильним?».

Отже, V-модель містить фази верифікації з одного боку та фази валідації з іншого боку, які об’єднуються фазою кодування у V-подібній формі, саме звідси і походить назва V-моделі [13].

Кожен етап моделі супроводжується контролем поточного прогресу, що дає можливість оцінити наскільки можливо та доцільно переходити на наступний рівень

проекту згідно із моделлю [14]. При цьому, ще зі стадії написання вимог починається процес тестування та продовжується для кожного наступного етапу відповідно до того, які очікувані результати та критерії входу/виходу прописано у сформованому тест-плані.

Як видно із пояснення, V-модель дотримується проходження через процес тестування на кожному етапу проектування і розробки системи відповідно до визначеного рівня. Тут процес розробки представлений низхідною послідовністю в лівій частині умовної букви V, а стадії тестування - на її правому ребрі. Відповідність етапів розробки та тестування показано горизонтальними лініями [14] (рис. 1.8).

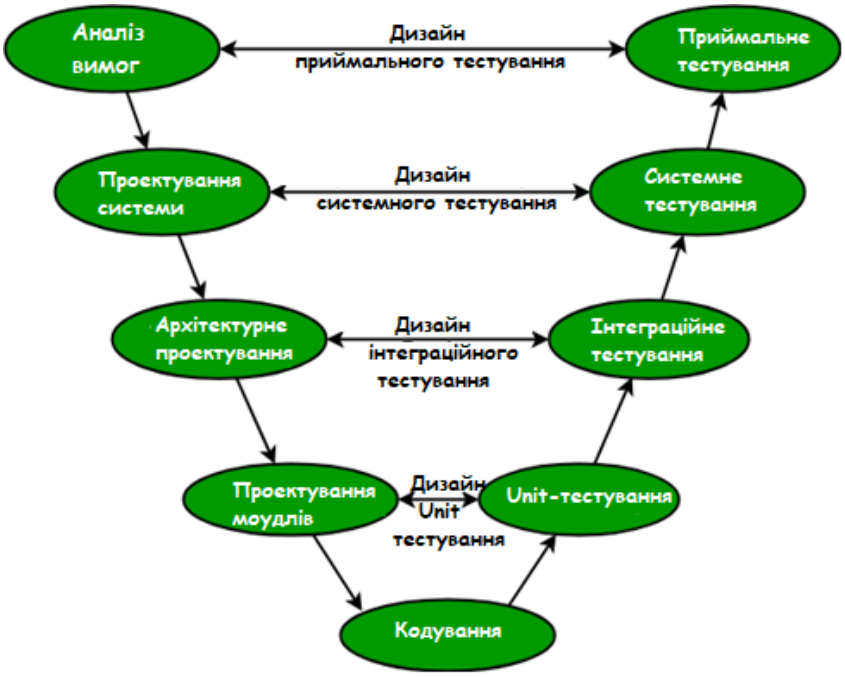

Рис. 1.8. V-модель життєвого циклу розробки ПЗ

Послідовні фази у моделі водоспаду описано у таблиці 1.4.

Таблиця 1.4

Фази V-моделі

Назва фази	**Діяльність фази**
Аналіз бізнес-вимог	- детальний зв'язок із замовником з метою визначення очікувань та точних вимог; - планування приймально-здавальних випробувань проекту.
Дизайн системи	- розробка структури системи (повна апаратурна та комунікаційна установка); - розробка плану тестування;
Архітектурний дизайн	- розробка архітектурних специфікацій (передача даних та зв'язок між модулями та зовнішніми системами); - розробка та документація інтеграційних тестів.
Проектування модулів	- визначення детального внутрішнього проекту для системних модулів; - розробка та документування модульних тестів;
Кодування	- фактичне кодування модулів
Тестування	- модульне тестування: тестування на рівні коду; - інтеграційне тестування: перевірка співіснування та взаємодії внутрішніх модулів; - системне тестування: перевірка всієї функціональності системи та зв'язку системи із зовнішніми системами; - прийомне тестування: перевірка продукту в середовищі користувача (виявлення проблем сумісності з іншими системами користувача, тестування навантаження).

V-модель виділяє наступні принципи розробки:

- ***Від великого до малого.*** Відповідно до цього принципу V-модель передбачає ієрархічне тестування, за якого вимоги спочатку визначаються командою проекту, потім в ході проектування високого та детального рівнів. Завдяки цьому, вимоги стають дедалі більш досконалими та деталізованими [11].

- ***Цілісність даних/процесів.*** За дотримання цього принципу елементи процесу визначаються з дотриманням кожної вимоги, що дає змогу досягти згуртованості всіх даних і процесів [12].
- ***Масштабованість.*** Концепція моделі передбачає гнучкість для розробки будь-якого ІТ-проекту (різний розмір, складність або тривалість).
- ***Перехресне посилання.*** Пряма кореляція між вимогами та тестами.
- ***Матеріальна документація.*** Кожен проект повинен створити документацію, яка використовується для ведення проекту.

Доцільність застосування V-моделі орієнтовно майже така ж, як у каскадної моделі, оскільки обидві моделі мають послідовний тип [12]. Можна виділити наступні сприятливі умови використання даної моделі:

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

1.2.3. Ітеративна розробка

Ітеративна модель розробки ПЗ – модель, яка передбачає поетапний процес, в якому кожна фаза додає функціональність до розроблюваного ПЗ та включає свій незалежний набір заходів з розробки та тестування [15]. Дана модель, по суті, є перехідною (*від каскадної до Agile*) моделлю розробки ПЗ і, на думку багатьох фахівців, оптимальною.

Такий підхід до розробки ПЗ виконується шляхом паралельного виконання робіт, що включає в себе процеси безперервного аналізу отриманих в ході цього результатів і коригуванням попередніх етапів роботи [15]. При цьому, ПЗ в кожній

фазі проходить через цикл Плануй-Роби-Перевіряй-Дій (Plan-Do-Check-Act - PDCA). Процес починається з простої реалізації невеликого набору вимог до ПЗ та ітеративного вдосконалення еволюціонуючих версій ПЗ, поки повна система не буде впроваджена і готова до розгортання [16]. Тобто, згідно основної ідеї цього методу, система розроблюється повторюваними циклами (ітеративно) та невеликими «порціями» за один раз (інкрементно).

Життєвий цикл розробки ПП розбивається на міні-цикли (ітерації) замість єдиної послідовності етапів, кожен з яких включає свої власні етапи аналізу вимог, проектування, реалізації і завершується тестуванням, інтеграцією та створенням працюючої частини системи [15]. Ітерація містить розробку окремого компонента(-ів) системи, який далі додається до вже існуючого функціоналу. Таким чином, система поступово збільшується крок за кроком і може приймати «товарний вигляд» за 10-15 ітерацій (рис. 1.9).

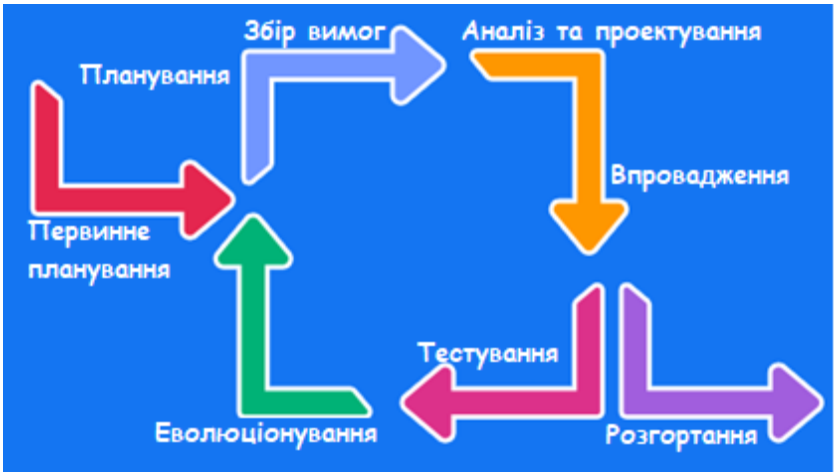

Рис. 1.9. Ітеративна розробка ПЗ

Послідовні фази ітераційної моделі описано у таблиці 1.5.

Таблиця 1.5

Фази ітераційної моделі

Назва фази	**Діяльність фази**
Планування	- складання специфікаційних документів; - загальної підготовки до майбутніх етапів циклу.

Продовження таблиці 1.5

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

Перевагою даної моделі, перед попередніми розглянутими традиційними, є те, що вона не вимагає повного обсягу вимог для початку розробки [16]. Вважається за достатнє – наявність вимог до базової частини функціоналу, а вже на наступних ітераціях, вимоги доповнюються, модифікуються і призводять до розширення функціоналу системи.

Від гнучких підходів ітераційну модель відрізняє те, що:

- вимоги до ПЗ розділені на кілька модулів, які можуть бути поступово розроблені та узгодженні. Тобто, спочатку розробляються основні функції, а все ПЗ розробляється шляхом додавання нових у наступні версії. В той час, як у Agile модель передбачає процес доставки, при якому поставляється кожна поступово розроблена частина, яке не обов’язково являє собою цілий компонент [15];

- у моделі ітеративного розвитку, як правило, немає фіксованого часу для завершення наступної ітерації, у Agile - дата завершення ітерації фіксована.

Ітеративний розвиток має деякі специфічні програми в індустрії ПЗ і найчастіше ця модель використовується за таких сценаріїв [16]:

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

1.2.4. Спіральна модель

Спіральна модель – модель розробки ПЗ, яка забезпечує підтримку управління ризиками, на схематичному відображенні дана модель відображається у вигляді спіралі (фази) із безліччю петель, кількість яких може варіюватися в залежності від проекту [17]:

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

Тому спіральна модель – модель, яка є подібною до «поступового» розвитку системи, який було приведено вище, але з більшим акцентом на аналізі ризиків [18]. Нижче на схемі показані різні фази спіральної моделі (рис.1.10) та розділення фаз моделі на квадранти (табл. 1.6).

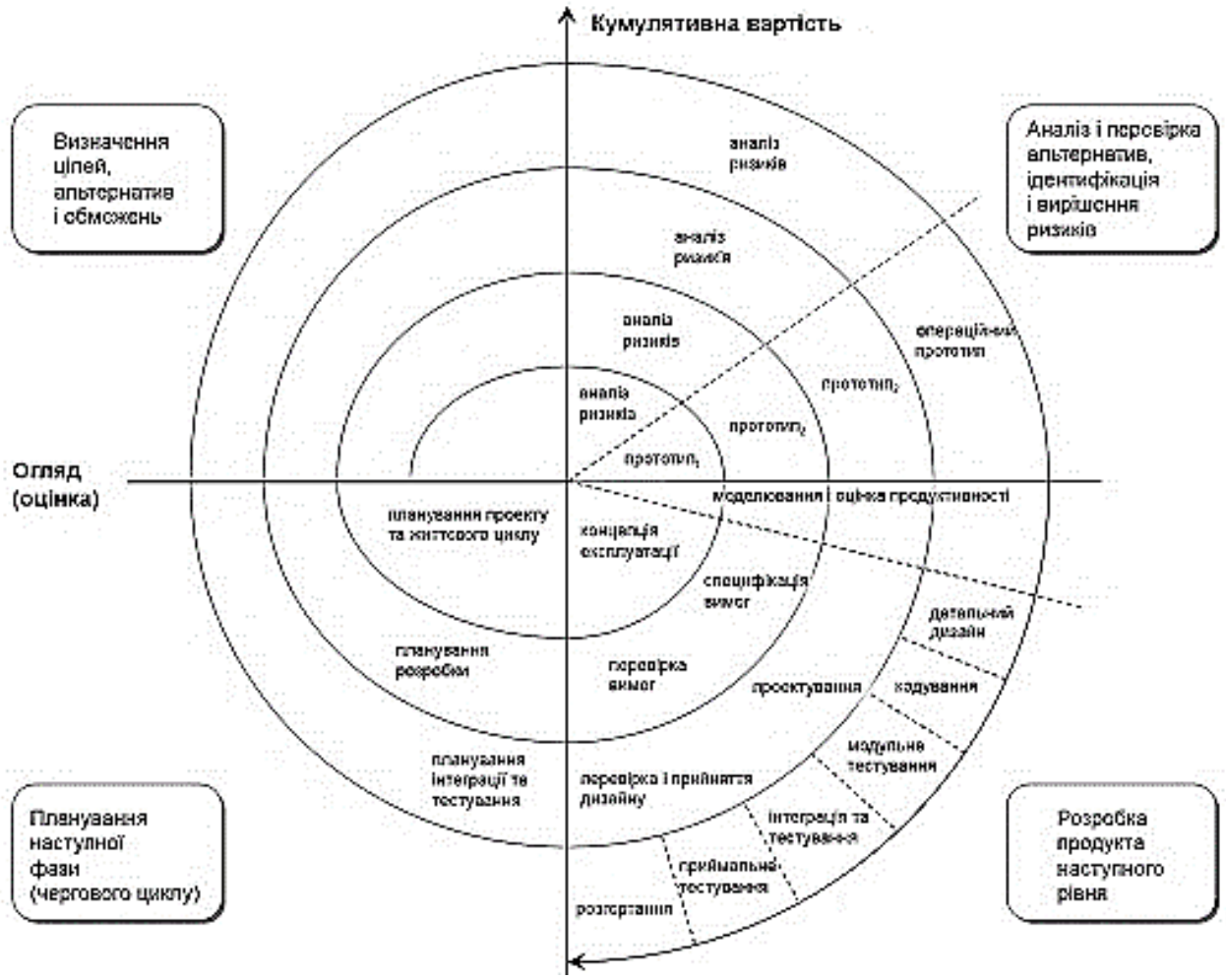

Рис. 1.10. Спіральна модель життєвого циклу ПЗ

Таблиця 1.6

Квадранти фаз спіральної моделі

Назва квадранту	**Функції квадранту**
Визначення цілей та альтернативних рішень	- збір вимог споживачів; - визначення, розробка та аналіз цілей; - визначення можливих альтернативних рішень.
Визначення та вирішення ризиків	- оцінка можливих рішень; - ідентифікація ризиків рішення; - розробка стратегії для ризиків; - розробка прототипу найкращого рішення.
Розробка версії продукту	- розробка та тестування визначеного функціоналу; - доступна версія ПЗ.
Огляд та планування наступного етапу	- оцінка розробленої версії ПЗ замовником; - планування наступного етапу.

Ризик, в даному випадку, розглядається як несприятлива ситуація і можливість її впливу на успішність завершення проекту ПЗ. І спіральна модель передбачає управління цими невідомими та несприятливими ситуаціями після початку проекту шляхом розробки їхнього «прототипу» на кожному етапі розробки ПЗ [18]. Існує ще таке поняття, як модель прототипування, що підтримує управління ризиками, але в цьому випадку вони повинні бути повністю визначені перед початком роботи. Проте, мінусом її є те, що, на жаль, в реальному житті ризик може виникнути вже після початку роботи, і в цьому випадку ми не можемо використовувати модель прототипування. Саме тому на кожному етапі спіральної моделі визначаються ризики та вирішуються шляхом створення їхніх прототипів. Таким чином, ця модель набагато гнучкіша порівняно з іншими наведеними [17].

Спіральну модель ще іноді носить іншу назву - мета-модель, внаслідок того, що вона включає у себе всі інші розглянуті вище моделі. Так, наприклад, спіраль з однією петлею насправді представляє ітеративну модель водоспаду, крім того, спіральну модель можна розглядати як підтримку ітераційної моделі, оскільки ітерації вздовж спіралі можна розглядати як еволюційні рівні, за допомогою яких будується повна система [18].

Отже, спіральна модель поєднує ідею ітеративного розвитку із систематичними, контрольованими аспектами моделі водоспаду, тому що є комбінацією ітеративної моделі процесу розвитку та моделі водоспаду з дуже високим акцентом на аналізі ризиків [18]. Завдяки цьому, існує можливість здійснювати поступову доставку продукту або поступове вдосконалення через кожну ітерацію навколо спіралі.

Спіральна модель є широко застосованою в індустрії ПЗ й наступні тези відображають типове використання спіральної моделі [18]:

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

ВИСНОВКИ ДО РОЗДІЛУ 1

Моделі життєвого циклу ПЗ за велику кількість років еволюціонування були удосконалені та розроблені нові. Все це робиться з тією метою, аби задовольнити величезну різноманітність й мінливість вимог, а також очікувань розвитку. У світі не існує такої моделі, яка б ідеально підійшла до усіх та будь-яких проектів, умов його старту та моделі інвестування, оплати. Навіть на перший погляд, багатоцільовий та універсальний Agile неможливо широко використовувати, внаслідок небажання деяких клієнтів й замовників масштабувати бюджет згідно із мінливих вимог.

Традиційні підходи використовують планування як свій механізм управління, тоді як моделі Agile використовують зворотний зв'язок від користувачів як основний механізм управління, тому цей підхід можна назвати більш орієнтованим на людей, ніж традиційні методи [2]. В порівнянні із плановими підходами, гнучкі доставляють робочу версію ПЗ набагато раніше, час «циклу» тестування порівняно короткий, внаслідок того, що здійснюється це паралельно до кодування, також вони є менш жорсткими і більш «гнучкі».

Отже, існує безліч варіантів моделей розробки ПЗ, але вибір того чи іншого варіанту залежить від особливостей і вимог проекту, моделей оплати. Звісно, частково методології перетинаються і схожі один на одного, але тим не менш, своїх шанувальників знаходить кожна із них, не дивлячись ні на що.

РОЗДІЛ 2
ПРОЦЕС ПОЄДНАННЯ ГНУЧКИХ І ПЛАНОВИХ ТЕХНОЛОГІЙ В УМОВАХ ГЛОБАЛЬНОЇ РОЗРОБКИ

Малі проекти та спільно розташовані команди більше не слугують обмеженням для впровадження гнучких технологій розробки ПП, спостерігається поширення масштабних та розподілених мереж команд, внаслідок чого Agile підхід став поширеними в організаціях, що займаються розробкою ПЗ в усьому світі. Дійсно, від початку свого існування дані методи застосовувались лише для розробки у малих та сконцентрованих в одному місці проектах, однак в останні роки багато великих організацій здійснили перехід від традиційних, орієнтованих на план методів водоспаду, до ітераційних методів розробки [19].

Виходячи із результатів опублікованих практичних досліджень можна припустити дві причини того, чому компанії застосовують гнучкі методи: одні вірять у цінності та принципи маніфесту, а інші вважають гнучкість найкращою практикою [20]. Але кожен проект повинен розуміти і усвідомлювати те, що не існує ідеального варіанту та треба знайти свою «золоту середину», що поєднує суміш традиційних та спритних методів.

2.1. Поняття «глобальної розробки» із залученням гнучких та традиційних підходів

Економічна ефективність, розширення кадрів висококваліфікованої робочої сили, виконання критичних завдань протягом всієї доби – фактори, які надає глобальна розробка, та які зробили великомасштабну розробку повсякденною реаль-

ністю в сучасних IT-організаціях. Переваги, дійсно, вражають, але не дивлячись на це, існує і низка складнощів в такій організації роботи, а саме: фізична відстань, різність часових поясів, відмінність культур, стратегічні проблеми, процесні відмінності, управління знаннями та технічні проблеми [19].

Слід розуміти, що такий термін, як «глобальна розробка» характеризується різно- національними та організаційними культурами зацікавлених сторін, які, до того ж, розташовані окремо в різних географічних місцях та часових поясах і використовують для співпраці різні інформаційні та комунікаційні технології. Як правило, такі умови можуть і часто призводять до виникнення проблем із координацією та співпрацею команди [19]. Крім того, ключові специфіки проекту, специфіка команди та відстань, конкретні контекстні фактори (характер договору, область застосування, мінливість вимог, персонал проекту, досвід роботи в команді та часова, географічна, соціокультурна відстань між партнерами) також можуть впливати на ефективність роботи команди.

Як зазначалося раніше, від початку свого існування гнучкі методології були спрямовані на невеликі проекти, однак все частіше великі організації прагнуть скористатися перевагами, які обіцяє такий вид розвитку в малому масштабі [2]. Гнучкість та пристосованість робочої сили – те, що так приваблює великомасштабну розробку та є основними вдосконаленнями важких до реалізації традиційних підходів до розробки ПЗ. Звичайно, з часом було запропоновано кілька практик та підходів до масштабування гнучких методі, таких як – Scrum-of-Scrum, LeSS, SAFe, DAD і т.д., але, незважаючи на безліч пропонованих підходів та фреймворків, існує недостатньо досліджень, що оцінюють їх вплив на галузі ПЗ [21]. Компанії власноруч намагаються комбінувати окремі елементи різноманітних практик, щоб задовольнити свої проектні потреби. Тобто, проглядається загальноприйнята практика змішування традиційних та гнучких підходів.

Різниця між підходами лежить у тому, що традиційні спрямовані більшою мірою на вирішення залежностей шляхом всебічного планування, детального визначення ролей та попереднього уточнення вимог до ПЗ та заходів із впровадження. В той час, як гнучкі підходи спираються на ідею, що розробники

можуть найбільше ефективно вирішувати залежності спільно та ітеративно шляхом самоорганізації [20]. Звідси випливає та підтверджується те, що традиційна великомасштабна розробка дотримується планового, структурованого підходу до виготовлення ПП, що передбачає розподіл завдань відповідно до того, де вони відображаються в життєвому циклі розробки ПЗ. В свою чергу, практики із залученням гнучких технологій вважаються здатними пом’якшити виклики, з якими стикається масштабна розробка [19]. Однак, складність таких великих проектів та потреба у залученні Agile технологій, означає, перехід організацій, для яких характерна розподілена розробка до зовсім нової практики, тому більшість схильні до прийняття та адаптування гнучких методів.

Взагалі, гнучкі технології були розроблені з метою підвищення продуктивності та ефективності розробки, залучення цих практик й справді покращує та підвищує задоволення користувачів з обох сторін – як клієнтів, так і розробників [19].

Отже, гнучкі технології розробки ПЗ відносяться до набору ітераційних та поступових методів інженерії, які виступають за "спритну філософію", відображену в маніфесті Agile, і їх можна вважати альтернативою традиційним методам розробки ПЗ. І якщо раніше вважалося, що гнучкі методи найкраще підходять для невеликих команд, розташованих разом, то їхній успіх у малих командах, надихнув їх використання у великих проектах, що було підтверджено багатьма масштабними фреймворками, які з'явилися за останні роки [20]. Але, все ж таки, фундаментальні припущення щодо гнучких підходів у межах великомасштабних проектів оскаржуються і є докази того, що вони можуть бути, навість, некорисними для великих підприємств, оскільки вимагають більшої координації та більш важких методологій, ніж менші.

Підсумувавши, треба сказати, що для традиційних методів розробки фундаментальною передумовою є ПП, який є повністю специфікованим і розроблений шляхом точного, всебічного планування. І, в той же час, концепція гнучкої розробки передбачає, що ПЗ можна будувати за допомогою поточного планування із подальшим вдосконаленням та тестуванням на основі швидкого зворотного зв'язку та змін [20].

Тобто, традиційні, іншими словами, планові підходи, архітектуру вважають за необхідне визначити ще до впровадження та тестування, тоді як гнучкий погляд на це передбачає архітектурний дизайн в результаті постійного уточнення в ході розробки. Архітектура є дуже важливим моментом при побудові великих програмних систем із залученням великої кількості команд із різних частин світу. І вона має бути узгоджена та затверджена заздалегідь, додаючи великого обсягу документації, бюрократії та накладних витрат [20]. Дійсно, глобальна розробка вимагає різних практик, коли справа стосується архітектурного дизайну, навіть при застосуванні гнучких методів.

2.2. Складнощі та виклики широкомасштабної гнучкої трансформації

Діджиталізація дуже стрімко змінила сферу послуг і на це вказують численні приклади в таких різноманітних галузях, як ЗМІ, фінансові послуги або роздрібна торгівля. Оскільки цифрові технології це така річ, яка піддається швидкому копіюванню, запозиченню або заміні, то конкурентні переваги тут недовговічні. Тому організації, що застосовували протягом десятиліть підходи, засновані на планах, мають впроваджувати гнучкі технологій, внаслідок того, що продукти, процеси та послуги є швидко змінними, прискоренням темпів технологічних змін, зміною поведінки споживачів та бізнес-моделей [20].

В цьому випадку постає складність у тому, що впровадження гнучких методів лише у сфери зайнятості розробки ПЗ може призвести до складного співіснування зайнятостей людей. І хоча вони традиційно практикувались у командах розробників ПЗ, зараз існує потреба у використанні їх для взаємодії з іншими організаційними підрозділами, такими як продажі, операції і т.д [19]. А це, у свою чергу, вимагає регулярного зв'язку між усіма підрозділами, аби впровадити процес BizDev (з метою постійної оцінки та вдосконалення координації). Це є не єдиним викликом, що виникає при впровадженні гнучких методів, оскільки сюди можна додати й пристосування до інших певних темпів поставок продукту клієнтові, іншої

діяльності з загального випуску та релізу продукції та нової моделі виплат винагороджень [19].

Проаналізувавши результати низки досліджень з цієї теми можна дійти наступних висновків: організації повинні визначити для себе певну сукупність тез для спрямованості на підвищення ефективності та гнучкості бізнесу, перш ніж впроваджувати Agile технології. Наступні процедури є ключовими (табл. 2.1). Згідно інформації, яка буде отримана згідно даної таблиці та подальшого аналізу, можна буде приділити увагу тим процесам, які знижують гнучкість [20].

Таблиця 2.1

Концептуальна основа для оцінки організаційної гнучкості

Сфера	**Процедура**
Стратегізація	Процес встановлення керівництвом амбіційної мети, розробки широкої загальнодоступної стратегії для керування та управління відданістю виконання.
Сприйняття	Процес регулярного моніторингу середовища з метою реєстрації змін настроїв робітників та звітування про це керівництву.
Тестування	Процес реєстрації експериментів, на яких організація створює, працює та навчається.
Впровадження	Процес реєстрації та звітування здатності й спроможності організації до впровадження змін.

Співіснування гнучких та планових підходів можливо, поєднання їхніх елементів ще й можуть доповнювати один одного. Але в реальному житті дуже часто таке співіснування спричиняє деякі, так званні, «напруженості» між командами, які вже впровадили гнучкі методи, та командами, які залишились при планових технологіях [19]. Такі складнощі можуть провокувати процеси бюджетування, знань, планування, процесу як такого, відповідальності та культури. «Напруженість», в даному випадку, можна визначити як елементи, які здаються

окремо логічними, але при їхньому зіставлені та поєднанні вони є суперечливими та навіть в деяких ситуаціях абсурдними. Тобто, в загальному сенсі напруженість виникає внаслідок двох контрастних, але суперечливих стилів управління.

Ці випадки свідчать про те, що ситуація, при якій попередньо використовуються планові методи і згодом приймаються гнучкі, є багатошаровим явищем, яке вимагає фундаментального переосмислення характеристик, практик та ролей [19].

Для вивчення та відображення впливу привласнення організацією обох структур на соціальну взаємодію можна спиратись на адаптивну теорію структурування (АТС) (рис.2.1). Завдяки їй можна дослідити взаємний вплив технологій та соціальних процесів, з метою аналізу пристосування груп людей до структур і впливу технологій на дані групи. Модель АТС складається з трьох елементів [20]:

- вхідна фаза - включає в себе джерела структури та внутрішню систему команди;
- процес - включає соціальну взаємодію між командами з різними джерелами структури. В свою чергу, соціальна взаємодія описує привласнення структур, які вже ведуть до раніше привласнених структур;
- вихідна фаза - включає результати проекту та нові соціальні структури.

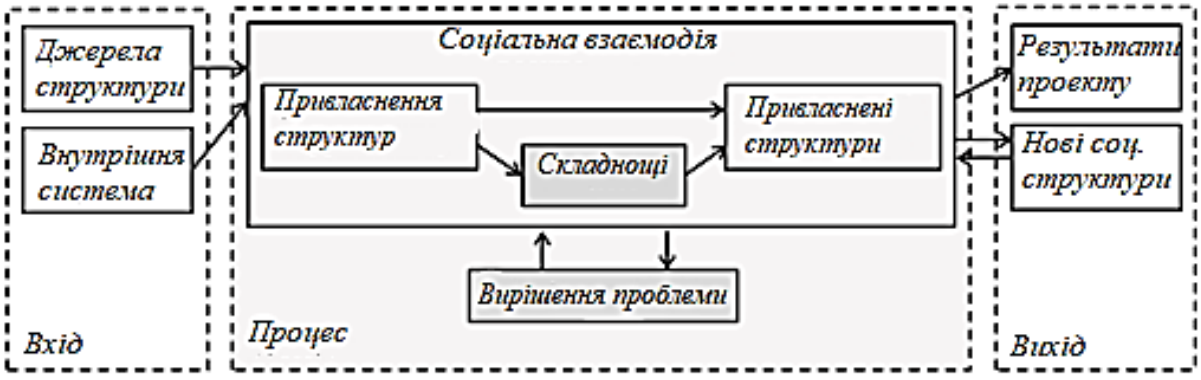

Рис. 2.1. Загальна АТС трансформації організацій

Розглянемо ситуацію, за якої джерелами структури будут планові та гнучкі

методи, а внутрішня система команди описує схеми взаємодії на рівні команд (наприклад, досвід співпраці членів команди разом, або з гнучкими методами, або їхній індивідуальний стиль керівництва і т.д.) [20]. Тобто, вхідна фаза включає керовані планами та гнучкі методи як джерела структури та внутрішньої системи.

Нехай до прийняття організація застосовувала методи, засновані на планах, «включаючи тривалий процес вимог та важку документацію», таким чином, структурні особливості методів, керованих планами, включають ієрархічну структуру для здійснення дисциплінарного контролю та послідовний план для дотримання суворої політики та процедур [21]. Потім компанія вирішує впровадити гнучкі методи відповідно до їхньої структури (а саме - усі команди відкриті для «швидкісних» методів і спрямовані на орієнтацію на клієнта, зосереджуючись на комунікації та ітераційних процесах). Що стосується внутрішньої системи – організація покладається на командно-адміністративне керівництво.

Фазу процесу розглядаємо як фокус для нашого аналізу. На цьому етапі ми описуємо «напруженість», що виникає внаслідок співіснування планомірних та гнучких методів (тобто джерел структури). Як соціальну взаємодію - координацію між командами з різними методами роботи (гнучкими та плановими), які генерують нові джерела структури і також впливають на соціальну взаємодію [20].

Нехай організація застосовувала метод водоспаду протягом десятиліть, відтепер, відповідні гнучкі методи були застосовані до загальних керівних принципів компанії. Окрім Agile команд, команди бізнесу та IT-операцій продовжуватимуть використовувати планові методи та цінні принципи водоспаду (наприклад, детальне планування та контроль проекту) [20]. Підсумовуємо можливі загальні «напруженості», що можуть виникати в ході трансформації (табл. 2.2).

Таблиця 2.2

Складнощі процесу трансформації

Сфера складнощів	Приклад проблеми	Підходи до вирішення	Тип вирішення
Бюджетування (Питання розподілу та контролю бюджетів)	*Наявність проблем виділення бюджету бізнес-командою конкретній команді через швидкісний та ітеративний підхід.*	- Пріоритетне бюджетування - Координаційна роль - Щотижнева зустріч з обміну	- Змішування - Балансування - Балансування
Знання (Відсутність навичок автономного завершення)	*Відсутність знань у менеджера щодо повного огляду процесу для запитів на послуги.*	- Координаційна роль - Щотижневі зустрічі з обміну	- Балансування - Змішування
Процес (Відсутність дотримання гнучких процесів)	*Клієнти розставляють пріоритети завданням, не консультуючись з власником нашого продукту.*	- Посилення автономії команди - Роль координатора - Посилення спритних ролей	- Змішування -Балансування - Змішування
Планування (Плановане планування протистоїть гнучким методам)	*Суперечки про фіксовані терміни з командою бізнесу через нерозуміння того, що неможливо детально спланувати продукт на рік вперед.*	*Дані відсутні.*	
Відповідальність (небажання та неможливість взяти на себе відповідальність)	*Заборона об'єднання коду у головну гілку без перевірки командою IT-операцій.*	- Реорганізація компанії - Спільна відповідальність	- Змішування - Змішування
Культурний (Культура, заснована на плані, стримує рухливі структури)	*Співпраця між бізнес-командою та agile-командою важка через різність точок зору.*	- Індивідуальні тренінги - Наймайння нового персоналу	- Змішування - Змішування

Отже, співпраця команд, як правило, викликає «напруженість», пов'язану з бюджетом, знаннями, процесами, плануванням, відповідальністю та культурою (див. табл. 2.2). Всі перелічені види «напруженостей» серйозно заважають процесу розробки отримання готового продукту. Крім того, фаза виходу виявила зміни у

показниках ефективності, таких як - якість, витрати та час (тобто результати проекту) [19].

Якщо обговорювати підходи до вирішення проблем, що стосуються виявленої «напруженості», можна сказати наступне. По-перше, було виявлено складнощі у бюджетуванні (див. табл. 2.2). Виходячи із планового підходу розподілу бюджетів, уся компанія за такого розкладу має боротися за розподіл одного бюджету для різних гнучких команд. Гнучкість розподілу бюджету для Agile команд можна підвищити, розподіляючи його залежно від пріоритету програмної функції, плюс до всього, додатково можна впровадити координаційного Scrum-майстра та проводити щотижневі зустрічі [20].

По-друге, було виявлено складнощі в обізнаності, або знаннях (див. табл. 2.2). Напруженість у знаннях можна вирішити шляхом регулярного обміну необхідними функціональними знаннями між новою гнучкою роллю та бізнес-командами, адже, обмін та розповсюдження знань є життєвоважливим двигуном для просування гнучких методів [20].

Що стосується складнощів, пов’язаних з процесом (див. табл. 2.2), можна ввести, так званні, змішані резолюції, посилюючи автономію команди та підтримуючи роль Scrum-майстра. Швидкі способи вирішення напруженості в процесі (наприклад, надання автономії гнучкій команді) сприяють Agile методам, підвищуючи їх статус на рівні команди [21].

Напруженість у відповідальності (див. табл. 2.2) може бути вирішена також змішаними резолюціями. Можна розпочати реорганізацію компанії шляхом об'єднання IT та бізнес-команд, інтегрувати розподілені знання та полегшити координацію між командами, які беруть участь у процесі розробки [19].

Культурна напруженість також визнається при застосуванні гнучких методів, оскільки при їх застосуванні протилежні спричиняють даний вид напруженості, який є особливо складним. Проте цей виклик можна також вирішити шляхом введення нових ролей та збільшення індивідуальних тренінгів.

Для позначення типів вирішення використано терміни «балансування» та «змішування» (табл. 2.2), щоб врахувати типи роздільної здатності [20]:

- «Балансування» означає *компроміс* між контрастними вимогами через механізми координації та ролі;
- «Змішування», в свою чергу, є гармонійним рішенням двох контрастних вимог для *поєднання* обох.

При цьому, резолюція вважається успішною, якщо вона сприяє прийняттю гнучких методів [20]. Згідно даної точки зору, балансування є більш направленим на впровадження механізмів координації (певні координаційні наради) або введення нових координаційних ролей (наприклад, є керівник процесу, який забезпечує постійний досвід споживачів), що пропонують лише проміжне рішення для напруги. Отже, резолюції балансу можна розглядати лише як проміжні кроки, що долають напруженість і, таким чином, відновлюють процес розвитку [20]. Таким чином, балансування напруженості, як правило, зберігає методи, зумовлені планами, тоді як змішування напруженості сприяє прийняттю гнучких методів. Узагальнення роздільної здатності було наведено вище (див. табл. 2.2), а процес розв'язання напружень - зображено на рисунку 2.2.

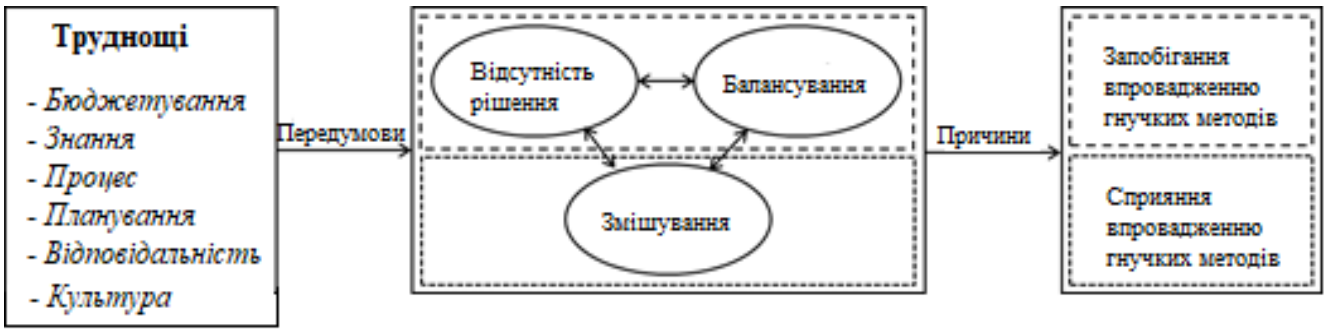

Рис. 2.2. Процес вирішення складнощів

Треба зазначити, що збалансовані резолюції, у першу чергу, перешкоджають застосуванню гнучких методів, а змішування резолюцій, навпаки, - сприяє гнучким методам та поширює спритні методи у відділи за межами розробки. Спираючись на АТС, було досліджено співіснування методів як соціальну взаємодію між гнучкими та плановими командами, завдяки чому, виявлено можливі складнощі між привласненням структур та впливом на них [19]. Отже, прийняття гнучких методів

й справді, як зазначалось вище, є багатошаровим явищем, що тягне за собою цілу реорганізацію структур, ролей та процесів, а забезпечення тісної співпраці між командами є неминучим процесом для досягнення повного переходу [19]. Таким чином, після усунення складнощів, які виникають внаслідок неправильної співпраці між гнучкими та керованими планами командами, організації змушені знаходити більш інтегровані, багатофункціональні підходи, наприклад, такі як DevOps, який можна розглядати як спосіб розширення гнучких методів та розбиття функціональних шахт [20].

Гнучкі команди-розробки ПЗ, що співпрацюють з іншими непровідними підрозділами, представляють виклик, оскільки гнучкі команди працюють в певному сенсі ітеративно, в той час, як інші одиниці можуть бути більш плановими та документ-орієнтованими [20]. Потреба саме гнучкоорієнтовних команд-розробників ПЗ для динамічної та оперативної взаємодії з іншими підрозділами організації – ось першопричина прагнення світових компаній розширити гнучкі методи поза відділи розробки ПЗ. Оскільки негайний і різкий перегляд усталених практик та швидкоплинний перехід до гнучких технологій рідко є можливим, то виникають гібридні підході, що несуть на меті поєднання обох методів. Емпіричні дані підкреслюють, що даний підхід мають два можливі витоки: або він успішно інтегрує обидва методи, або ж призводить до їхньої складної взаємодії та співіснування.

Отримані дані відображають, що врегулювання «напруженості» змушує компанії визнавати гібридні методи (тобто вигідну комбінацію планових та гнучких методі) як потенційне рішення або застосовувати інші гнучкі методи [21].

2.3. Гібридна розробка у межах великомасштабних проектів

Глобалізація, яка є однією із тенденцій XXI століття, внесла суттєві та відчутні зміни у велику частку галузей, включаючи і розробку ПЗ, зокрема. Багато компаній з метою отримання вигоди від дешевшого, швидшого та кращого, у всіх розумінях, процесу розробки ПЗ сприяють поняттю глобальної розробки. І якщо у минулому

середовище допускало використання планових методів, які включають в себе масштабне та трудомістке попереднє планування проектів (наприклад, розробка вимог та специфікація і т.д.), а отже - тривалий період реалізації, то на сьогоднішній день переважними конкурентними показниками є прискорені можливості розробки продуктів та висока операційна гнучкість [20]. Таким чином, сьогодні ми можемо спостерігати, як зростає кількість підприємств у різних галузях, що використовують методи, засновані на планах, при цьому застосовуючи гнучкі методи, а як вже було згадано, дана комбінація називається гібридними підходом.

Гібридний підхід до розробки ПЗ - це будь-яке поєднання гнучких та традиційних (орієнтованих на план) підходів, які організаційний підрозділ використовує та підлаштовує під свої власні контекстні потреби (наприклад, домен програми, культура, процес, проект, організаційна структура, методи, технології тощо) [21]. Іншими словами, гібридні підходи до розробки ПЗ охоплюють різні елементи керованих планами методів (наприклад, водоспад або V-модель) та гнучких методів (наприклад, Scrum або Kanban).

Ключовими характеристиками даного підходу є такі, що демонструють [19]:

- наявність характеристик двох або більше підходів (наприклад, ієрархічне спілкування зверху вниз за традиційним підходом в поєднанні з корекцією знизу вгору, що вбачаються в гнучкому підході);
- Наявність практик двох або більше різних підходів (наприклад, широкомасштабне попереднє планування в традиційному підході в поєднанні з регулярними перевірками та адаптацією гнучкого підходу);
- Наявність ролей з двох або більше різних підходів (наприклад, традиційні менеджери проектів, Scrum-майстри та власники продуктів гнучких підходу).

Таким чином, організаціям вдається досягнути специфічний гібридний підхід завдяки континууму між обома методами (рис.2.3), навмисно вибираючи як гнучкі, так і керовані планами елементи для поєднання корисних елементів [20].

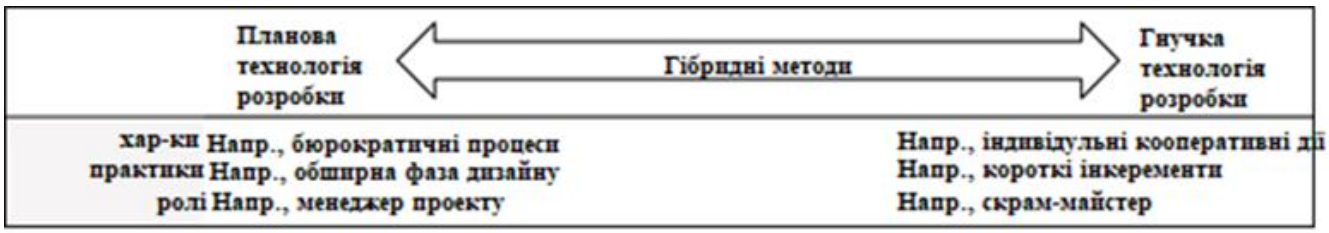

Рис. 2.3. Гібридний підхід шляхом континууму між методами

Аби відповідати складній природі великих проектів, що включає численні команди розробників, фундаментальні структури та методи маломасштабної гнучкої розробки ПЗ піддаються адаптації. Було висловлено припущення, що гібридні підходи, що поєднують традиційні планові та сучасні гнучкі розробки, насправді часто є адекватним вибором, враховуючи історію та вимоги великих, тривалих та високо інтегрованих проектів розвитку [20]. Але, все ж таки, передбачуваність, надійність, стабільність та ефективне використання ресурсів – е першопричини, які без остраху можна наведені на користь продовження традиційного планування в таких гібридних підходах.

Деякі дослідники стверджують, що поєднання суміші традиційних і спритних практик є найкращим способом управління проектом, оскільки методи повинні бути адаптовані до потреб робочого контексту протягом життєвого циклу програми, і хоча є багато корисних порад застосування гнучких технологій у великомасштабних проектах (Agile Framework та масштабний Scrum), вони врешті-решт обмежують організації [21].

Для подальшого аналізу набору методів та можливостей їхнього комбінування використаємо три категорії - традиційний, гнучкий, і загальний, де [21]:

- Agile + Загальний = Agile;
- Традиційний + Загальний = Традиційний;
- Agile + Градиційний = Гібрид.

При цьому, як гнучкі будемо розглядати та класифікувати наступні підходи: Scrum, Safe, Lean, LESS, Nexus, XP, Kanban, DevOps, ScrumBam, Crystal, DSDM та FDD. Аналогічно, як традиційні: водоспад, спіральна модель, V-модель, RUP, PRINCE2 та SSADM. Інші підходи будемо класифікувати як загальні (на основі

того, що дані підходи не вкладають в жоден спринт чи традиційну категорію): ітеративна розробка, дизайн доменних дисків (DDD), архітектура, керована моделями (MDA), командний процес ПЗ (TSP), процес персонального ПЗ (PSP) [21]. Ці правила ґрунтуються на зауваженні, що "загальні" процеси можуть бути або гнучкими, або традиційними - залежно від контексту, що в цьому випадку визначається іншими використовуваними методами, які є спритними або традиційними. Виходячи із результатів кількісних та якісних опублікованих досліджень, на сьогоднішній день можна зробити наступний розподіл проектів за категоріями (табл.2.3) [19].

Таблиця 2.3

Розподіл проектів за категорією

Клас категорії	**%**
Agile	25.0
Традиційний	2.0
Гібридний	72.0
Інший	1.0

Дане дослідження було проведено на основі 263 обґрунтованих та повних відповідей, пов'язаних з проектами глобальним проектами ПЗ, участь взяли 33 різних країни, що охоплювали 5 континентів [19].

На рис. 2.4 представлені фреймворки гнучких технологій та ступінь їхнього використання в межах глобальних проектів. Використовуючи наведені дані, можна дійти висновку, що серед гнучких підходів найчастіше використовуються Scrum та Kanban, а найрозповсюдженішим традиційним підходом є водоспад (waterfall). Тобто за використання гібридного підходу, проект може використовувати, наприклад, Scrum, але поєднувати додатково деякі методи, орієнтовані на плани. Також, глобальні проекти використовують більше підходів ітеративних розробок та підходів на основі Канбану. Серед фреймворків, які були створені для

масштабування гнучкості, за використанням переважають LESS та Safe, що випливає з Nexus [19].

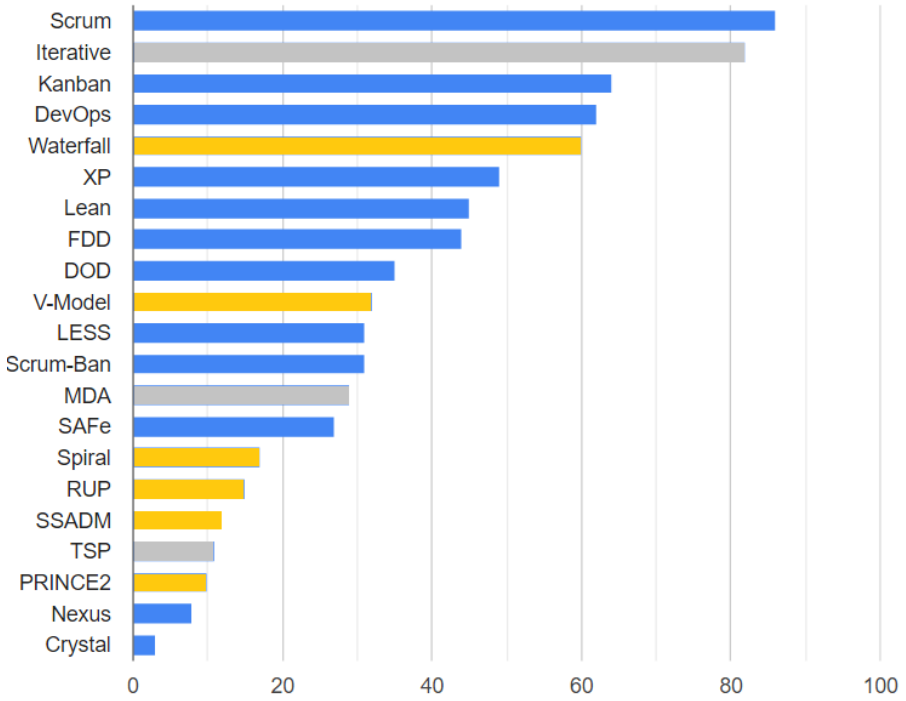

Рис. 2.4. Частота використання підходів (гнучкий – синій, традиційний - помаранчевий, загальний - сірий)

Що ж зумовлює команди здійснити вибір у користь того чи іншого підходу? Згідно практики, вивчення великої кількості досліджень, а також статей, та підсумувавши, можна виокремити наступні цілі, що спонукають до певного вибору [19]:

- підвищення робочої продуктивності;
- покращення процесу планування та оцінки;
- підвищення якості зовнішньої продукції;
- збільшення частоти доставок клієнтові.

Отже, в залежності від очікувань та вимог, кожна організація може знайти для себе те, що їй необхідно, тим паче, що на сьогоднішній день велике поширення мають гібридні підходи. Реалізація глобальних гібридних проектів - це лише комбінація обраних підходів, що працюють разом, щоб підвищити ефективність, що перевищує будь-яку єдину методологію [20].

При цьому гібридний підхід має свій ряд переваг, таких як: належна підтримка бізнес-проектів (наприклад, стосовно внутрішніх цілей клієнта), допомагає досягти хорошої якості продукції, адекватно відповідає зовнішнім вимогам (наприклад, стандартам) та допомагає пришвидшити розробку й скоротити час виходу продукту на ринок, а також дає можливість постійно вдосконалювати якість завдяки гібридним конструкціям, покращити задоволеність команди [19].

Проте, слід мати на увазі, що використання «ексклюзивних» і спеціальних методів та практик не обов'язково призводить до більш компетентної команди розробників і, навіть, може підірвати якість та продуктивність ПЗ, а також соціальні фактори, такі як мотивація та ефективність роботи команди. Таким чином, крім поєднання методів для кращих результатів продукції, необхідно структурувати філософію для гібридних команд, що може покращити задоволеність команди [19].

Для того, аби порівняти як часто традиційні та гнучкі методи поєднуються в проектах великого масштабу, результати було зведено до таблиці 2.4., де рядки показують конкретні гнучкі підходи, а стовпці показують частоту кожного традиційного підходу.

Таблиця 2.4

Частота комбінацій гнучких та традиційних методів (де 1 – поширено, 2 – помірно)

	PRINCE2		RUP		Spiral		Waterfall		V-Model	
	1	2	1	2	1	2	1	2	1	2
Scrum (86%)	3%	8%	4%	13%	4%	14%	18%	41%	14%	19%
Kanban (64%)	3%	7%	3%	13%	3%	12%	19%	40%	17%	20%
Dev Ops (62%)	3%	9%	4%	11%	1%	14%	18%	38%	9%	19%
XP (49%)	2%	8%	6%	18%	2%	21%	17%	46%	9%	25%
Lean (45%)	4%	9%	6%	15%	6%	18%	17%	45%	20%	22%
FDD (44%)	3%	10%	3%	14%	7%	16%	22%	43%	20%	21%
LESS (31%)	4%	10%	7%	18%	4%	17%	18%	48%	13%	26%
Scrumban(31%)	4%	10%	6%	17%	4%	19%	22%	41%	15%	28%
SAFe (27%)	4%	15%	10%	22%	6%	21%	22%	51%	18%	25%
Nexus (8%)	10%	30%	10%	40%	5%	30%	15%	40%	10%	45%
Crystal (3%)	14%	43%	29%	43%	14%	43%	14%	43%	29%	43%

Згідно результатів, гібридні проекти легше знайти, ніж суто гнучкі. Зрозуміло, що можливості адаптації та налаштування кожного з цих підходів повинні

відповідати конкретним потребам сучасної ситуації, щоб надати організаціям можливість розробити проект із залученням гнучких технологій та використовувати найкращі методи у певному аспекті роботи [20].

Якщо розглядати це питання з точки зору того, для яких стандартних процедур та заходів при розробці ПП надається перевага, то можна отримати наступний графік (рис. 2.5), де [19]:

1 – повністю традиційний підхід;

2 – в більшості своїй традиційний;

3 – збалансоване використання традиційного та гнучкого підходу;

4 – переважно гнучкий підхід;

5 – повністю гнучкий.

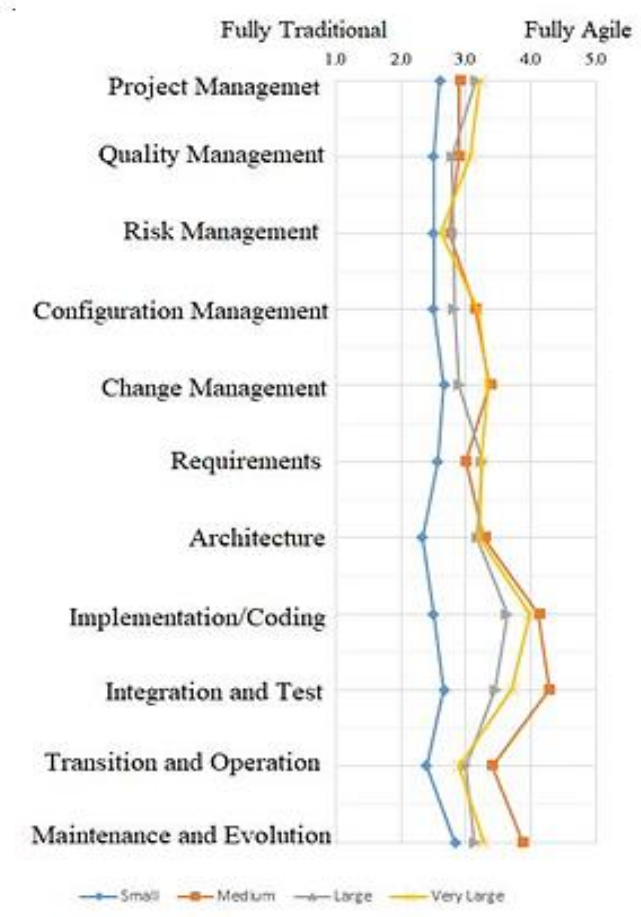

Рис. 2.5. Впровадження заходів у проектах глобальних масштабів

Як видно із наведеного графіку, загалом, все вказує на те, що в більшості випадків діяльність проводиться збалансовано між гнучкими та традиційними методами. Однак такі заходи управління, як управління ризиками та управління якістю, є все ж таки більш традиційно орієнтованими, тоді як заходи з розвитку, такі як впровадження/кодування та інтеграція/тестування, як правило, проводяться більш дотримуючись гнучких підходів [21]. Таким чином, кілька «гнучких» артефактів, такі як історії користувачів та беклог, використовуються у глобальних проектах, крім того, команди переймають елементи гнучкої культури [20]. Однак, для вирішення датування релізу, тестування та впровадження архітектури продукту, прийнята методологія у більшості випадків відповідає плановому підходу.

Основними причинами зацікавленості застосування гнучких методів у межах глобальної розробки ПЗ є вирішення таких питань (або наближення до їхнього вирішення), як складність розробки ПЗ та посилення уваги на проблемі координації, спілкування та співпраці. Хоча Agile підхід має багато переваг, наші результати свідчать про те, що він все ж таки не є універсальним рішенням, і як результат, ми часто бачимо компанії, що використовують поєднання гнучких та планоорієнтовних методів, що називається гібридним підходом. Він є результатом компромісу використання переваг ітеративного, логічного та спільного підходів із підтримкою певного рівня планування та структури [19]. Як видно із результатів розділу, на практиці гібридні підходи прийняті в більшості (72%) проектів глобальної розробки, тоді як 25% використовують лише спритні методи, а 5% - лише традиційні.

ВИСНОВКИ ДО РОЗДІЛУ 2

Отже, розробка ПЗ на глобальному рівні і в великих масштабах не є перешкодою для прийняття гнучких практик. Багато хто для полегшення координації проектів та загального управління ними застосовує ще й традиційні підходи, в результаті чого, описаний вище, - гібридний підхід, який відповідає визначеним правилам. Згідно статистики, більшість глобальних програм не є ні

гнучкими, ні чисто традиційними у своєму підході до розробки ПЗ, а скоріше поєднують зазначені два методи.

Не дивлячись на плюси адаптованого методу, гібридний підхід має і несприятливі наслідки, такі як - проблеми для продуктивності проекту та затримки випусків. При чому, їхні масштаби варіюються в залежності від створення вимог та узгодження методів до різних думок щодо процесів та стандартів. Зокрема, адаптація та узгодження методів та практик розробки ПЗ в середовищі між двома протилежними підходами до розробки є серйозними труднощами для організацій [20].

Підсумовуючи, гібридні підходи мають численні потенційні переваги, накладаючи різноманітні виклики та складнощі, тому дуже важливою є необхідність координації між усіма командами. Однак, на жаль, ще не з’ясовано причини, звідки випливають часто згадані проблеми координації в гібридних установках та чому вони виникають. З цієї причини в наступних розділах надається побудова теорії, щоб створити пояснювальну основу для подальших досліджень [19].

РОЗДІЛ 3

ПРОЦЕСИ ВІДБОРУ ТА РЕІНЖИНІРИНГУ АРХІТЕКТУРИ В ІТЕРАЦІЯХ ГНУЧКИХ МЕТОДІВ ЯК ВАЖІЛІ КООРДИНАЦІЇ

Суттєві рішення про організацію програмного комплексу, вибір структурних елементів і інтерфейсів системи, взаємодія елементів та їхня інтеграція – всі ці питання охоплює архітектура ПЗ.

Таким чином, архітектура ПП є скелетом системи, навколо якої обертаються всі інші аспекти системи [22]. Відповідно, підвищення загальної гнучкості, визначення легкості внесення змін до системи, організаційні аспекти керівництва, план розробки системи залежать від архітектури системи. У гнучких технологіях розподіл роботи, зазвичай, виконується без визначеної архітектури, що становить собою причину і привид для критики великомасштабної гнучкої розробки. Оскільки дана ситуація може призвести до редизайну, дивергенції архітектури та функціональної надмірності, збільшуючи, таким чином, складність системи.

3.1. Поняття гнучкої архітектури в межах адаптованого Agile

Даний підрозділ у якості адаптованого методу гнучких технологій розглядає Scrum як один із найпопулярніших. Адаптований Scrum – це спеціальний процес розробки ПЗ, в якому ключовим видом діяльності під час підготовки спринтів є гнучка архітектура, яка систематично використовується як в великих, так і в малих ПП згідно відповідних механізмів [22].

В той час, як в проектах із Scrum-ом є основними наступні ролі – Scrum-майстер, власник продукту, команда Scrum. У випадку адаптованого проекту

необхідне включення ще і ролей архітекторів, які взаємодіють з іншими членами гнучкої команди у процесі прийняття рішень шляхом відстеження проблем у архітектурі та збалансування їх з бізнес-пріоритетами [22] (рис. 3.1).

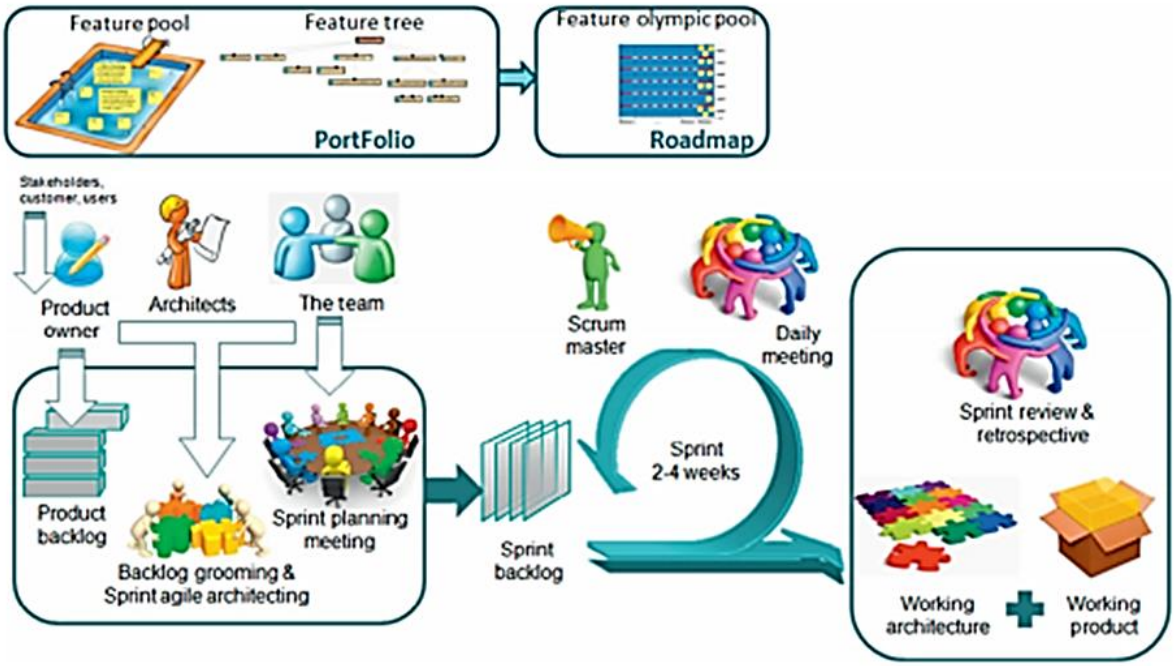

Рис. 3.1. Гнучка архітектура в адаптованому Scrum

Тобто, включення архітекторів у ланцюжок полегшує координацію та обмін знаннями між членами команди.

Процес адаптованого Scrum починається із визначення портфоліо (набір властивостей і функцій) і дорожньої карти проекту (встановлення дат), які є фундаментом для гнучкого масштабування [22]. Дорожня карта передбачає також поняття Olympic pool, яке представляє собою базу беклогу продукту з історій користувачів та приймальних тестів. Кожен спринт включає в себе такі механізми координації, як огляд беклогу, архітектурні збори та зустріч планування спринту, завдяки яким історії набувають пріоритетів та підлягають розподілу по спринтам аналогічно класичному Scrum-у.

Під час огляду беклогу дуже допоміжним та важливим для розуміння робочого продукту, подальшого вивчення, планування та пріоритезації є роль архітектури ПЗ [24]. Такі сеанси огляду є ідеальними для проведення аналізу

робочої архітектури, що дає можливість визначити вплив різних рішень наступних спринтів і побудувати певні критерії.

Проведена пріоритезація історій користувачів також має важливе значення для гнучкої архітектури, оскільки архітектура будується виходячи із обраних для реалізації функцій [22].

Зустрічі із планування спринту формують беклог спринту, протягом якого проводяться щоденні мітинги відстеження прогресу команди над продуктом, опираючись саме на його архітектуру [23].

3.2. Рефакторинг архітектури як складова ітерацій проектування та забезпечення якості продукту

Мінливість та введення нових вимог, редагування інфраструктури й технологій – зміни, які породжують виникнення помилок та невірних рішень і в таких випадках важливим є зосередитись на архітектурі ПЗ [24]. Оскільки у випадку неконтрольованого додавання нового функціоналу або остаточно прийнятого невірного рішення, проект почне збільшуватися до тих пір, поки стане некерованим та непідвладним, що унеможливить його супроводжуваність.

У зв’язку із цим, архітектура має редагуватися лише шляхом ітеративної оцінки та рефакторингу, тобто шляхом поліпшення структури без змін у зовнішній поведінці системи [23]. Безперервний приріст та еволюція архітектури ПЗ є потенційною областю для рефакторингу, що підкреслює необхідність впровадження оцінки архітектури та рефакторингу на всіх ітераціях (рис. 3.2).

Рис. 3.2. Алгоритм оцінки та рефакторингу архітектури кожної ітерації

Рефакторинг включає в себе, так звану, область ідентифікації, яка є індикатором визначення проблем в архітектурі [22] і потенційно має її поліпшити (табл. 3.1).

Таблиця. 3.1

Поширені області ідентифікації

Область ідентифікації	**Опис**
Дублювання артефактів проекту	Присвоєння різним компонентам архітектури одних і тих самих обов'язків. Складність полягає у визначення допустимого обсягу реплікацій. В іншому випадку, страждає принцип DRY (неповторюваності).
Нечіткість ролей сутностей організації	Назви компонентів повинні пояснювати їхні властивості. При цьому, кожен компонент повинен мати лише один обов'язок, кожен обов'язок повинен бути призначений окремому компоненту. Інакше, страждають принципи поділу проблем.

Продовження таблиці 3.1

Складність архітектури	Випадкова складність призводить до непотрібних абстракцій, що веде до виникнення невиразних ПП і незрозумілої архітектури. Сюди можна віднести присвоєння незрозумілих імен об’єктам, надлишковість компонентів, недостатню/надлишкову деталізацію
Нові самостійні рішення замість загальних практик	Винаходження колеса, замість використання перевірених готових рішень, які часто перевершують самостійні рішення.
Сверхуніверсальність проекту	Зловживання патернами може призвести до страждання переносимості та виразності, важкості формування, подальшого розвитку та підтримки архітектури. Проектування архітектури ПП має бути універсальним та конфігурованим в міру необхідності.
Асиметрична структура та поведінка	Симетрія архітектури (поведінкова та структурна) є показником високого рівня міжмережевої взаємодії, асиметрія – може вказувати на потенційні архітектурні проблеми. (А)симетрія не обов'язково є доказом гарної чи поганої архітектури.
Цикли залежностей	Існування циклів залежностей між архітектурними компонентами свідчить про насування проблем. Це може чинити негативний вплив на процеси тестування, модифікацію та виразність.

Продовження таблиці 3.1

Проектні порушення	Порушення політики проектування, наприклад, розробка деякого нашарування замість суворої ієрархії. Інакше, це може зменшити видимість і виразність через неконтрольовані рішення різними інженерами.
Невідповідний поділ функціональності	Складові підсистеми повинні мати високу єдність, а зв'язок між ними - низьку. Інакше, це може бути неправильне розбиття функціоналу на підсистеми. Рішення може полягати у підсиленні зчеплення або метрики єдності.
Непотрібні залежності	Надлишковість залежностей може бути вирішена шляхом простого скорочення додаткових і непотрібних (випадкових) залежностей, оскільки вони можуть впливати на продуктивність та змінність.
Неявні залежності	Наявність залежностей у ПП, але їхня відсутність в архітектурних моделях, може призводити до багатьох проблем. Це погрожує порушення реалізації.

Слід зауважити, що наведені області ідентифікації, не завжди є прямим доказом проблем в архітектурі, вони можуть просто на можливе існування таких проблем [23]. Визначається це конкретним контекстом проблеми та особливостями ПП.

Вирізняють два види якості архітектури ПЗ [22]:

- внутрішня якість, яка характеризує структурні аспекти (симетрія, залежності і згуртованість) і вимірюється метриками та інструментами оцінки архітектури;

- зовнішня якість, яка характеризується згідно атрибутів якості стандарту ISO/IEC 25010.

Рефакторинг проводиться завжди із метою поліпшення характеристик розробки та покращення якості ПП, тому архітекторам необхідно адресувати дві різні області: показники якості архітектури (внутрішня якість) і атрибути якості (зовнішня якість), також аналіз впливу внутрішніх якостей на зовнішні [22]. Розглянуті вище області ідентифікації допоможуть у виявлені можливих проблем.

Усі дії з рефакторингу повинні проводитися ітеративно і систематичним чином згідно з алгоритму, зазначеного вище (див. рис. 3.2). Нижче наведено наглядну схему процесу [23] (рис. 3.3):

Рис. 3.3. Ітеративна схема процесу рефакторингу

1. Детермінація необхідності проведення рефакторингу – процес визначення неякісно написаного коду і проблем проектування. Як результаті отримується список ідентифікованих архітектурних проблем [23].

2. Пріоритезація – процес розміщення за важливістю всіх визначених проблем архітектури, шляхом аналізу пов’язаних вимог. В

результаті отримується розподіл проблем згідно їхнього пріоритету та обсягів. розподіліть проблеми архітектури щодо їх пріоритетів і обсягів [23].

3. Вибір способу резолюції – процес, за якого кожній визначеній проблемі виконується одна з наступних дій [22]:

- обираються відповідні патерни для проведення рефакторингу;
- за наявності декількох підходящих шаблонів, обирається той, впливи якого є більш відповідним;
- якщо шаблонів не існує, то відбувається звичайне перепроектування.

Ідея рефакторингу з патернами була запропонована Джошуа Керівським і головна мета полягає у заміні власностворених рішень шаблонами для конкретної проблеми [24]. Вагомий аргумент полягає в тому, що нема потреби вручну прописувати посилання на події, шаблон самостійно будує динамічні залежності з важливими подіями, переносячи акцент з кодування на проектування.

3.3. Проектування архітектури на основі типових рішень із врахуванням вимог якості

Під архітектурним стилем розуміють набір принципів, які в сукупності представляють собою високорівневу схему, яка забезпечує інфраструктуру для програмного комплексу; це є деяке сімейство систем, утворене на основі організації структури [22]. Він дає змогу покращити секціонування елементів та сприяє можливості повторного використання дизайну, завдяки тому, що забезпечується рішення найрозповсюдженіших проблем.

В даному контексті, патерни проектування є описом взаємодії об’єктів та класів для розв’язання загальної задачі проектування у межах конкретної проблеми/задачі [23]. Загалом патерни проектування розділяють на три класи – породжуючі, структурні та поведінкові.

Таким чином, кожна система будується на основі, як мінімум, одного такого патерну (звісно, як правило із значно більшої кількості, в залежності від її складності та масштабності) і реалізує при цьому один з архітектурних стилів (можна розглядати як деякий каркас) [22]. Наприклад, взявши такі патерни, як Команда, Спостерігач, Стан, Фасад і т.д., можна утворити систему, яка буде відповідати клієнт-серверному архітектурному стилю.

Згідно класифікації та уніфікації характеристик якості архітектурних стилів розрізняють наступні моделі (стилі) архітектури: клієнт/сервер, компонентна архітектура, проектування на основі предметної області, багатошарова архітектура, архітектура на основі шини повідомлень, N/3-рівнева архітектура, об'єктно-орієнтована архітектура, сервісно-орієнтована архітектура тощо [23].

Кожен архітектурний стиль має власний властивий йому набір характеристик якості, але таке трактування не є очевидним через те, що кожна характеристика якості архітектури може бути пов'язана як одна-декілька стандартних (під-)характеристик якості ПП. Від початку архітектурні моделі були запропоновані без врахування показників якості самої архітектури кожного відповідного стилю і вимоги якості через це вводяться «наздогін» [22]. Тобто виконується спроба використати ті чи інші характеристики якості, але кожен розробник може використовувати власний набір цих характеристик [22].

Існує гостра потреба у існуванні та створенні уніфікованої моделі характеристик якості архітектури ПЗ для порівняння різних стилів, причому ці характеристики повинні бути безпосередньо пов'язані з характеристиками якості ПП.

3.4. Компонентний підхід до розробки програмної архітектури

Проектування програмної архітектури можливе за використання, так званого, компонентного підходу. На сьогоднішній день цей погляд є ключовим, внаслідок того, що більша частка задач, які реалізуються за участі ПЗ, є типовими і потребують лише певної адаптації до визначеної предметної області [25]. Це

дозволяє створювати нові ПП без повторної розробки його окремих елементів, а із застосуванням вже готових рішень.

У рамках програмної архітектури, під поняттям компоненту розглядається деякий програмний модуль, що є досить абстрактним елементом і вирішує певні покладені на нього задачі в рамках спільних завдань системи і взаємодіє з оточенням через інтерфейс [26]. Тут робиться акцент на можливості виділення структурних елементів системи і можливості декомпозиції вирішуваних нею задач.

Окрім цього «компоненту», існують ще інші у рамках UML-проектування та компонентної розробки ПЗ. На діаграмах UML компонент являє собою одиницю збірки або конфігурацію, які можуть являти собою файли з кодом, бінарні файли, документи системи [25]. Дане поняття розглядає структурні елементи системи, з якими співпрацюють додатки для створення збірок та конфігураційного управління.

У рамках компонентного програмування, компонент розгортання є блоком для побудови ПЗ і являє собою одиницю системи. Як правило, такі компоненти розгортання є елементами компонентних технологій, компонентно-орієнтованої (component based) розробки, або можуть виступати як компоненти JavaBeans, EJB, CORBA, ActiveX, VBA, COM, DCOM, .Net, Web-служби (web services) [25]. В цьому випадку, під компонентом розглядається виділена одиниця з визначеним одним або декількома інтерфейсами, де прописані всі залежності.

Плутанина або неповне усвідомлення різниці цих понять може призвести до серйозного непорозуміння.

Компонентний підхід на етапі розробки програмної архітектури дає змогу впровадити процес контролю якості, з огляду на те, що архітектура сама по собі є набором компонентів реалізації функціоналу ПЗ та зв’язків між ними [27]. Кожен з яких має свій, властивий йому, набір функціональних та нефункціональних характеристик якості. Компонент може бути розглянутий як самостійний продукт (складений із одного і більше об’єктів логічного рівня проектування), що реалізує окрему предметну область і може взаємодіяти з іншими через інтерфейси [28].

Звідси випливає, що архітектура системи є абстрактною моделлю-каркасом, яка поєднує набір взаємодіючих об’єктів у інтегроване компонентне середовище,

ціллю якого є досягнення певної кінцевої мети. Така архітектура може складатися з [25]:

- компонентів-серверів;
- компонентів-контейнерів;
- внутрішньо-контейнерних реалізацій функцій;
- реалізацій компонентних моделей, об'єктів, що задовольняють установку і конфігурування окремих компонентів для деякої комп'ютерної платформи [25];
- клієнтських компонентів (веб-клієнтів, графічного інтерфейсу) та інтерфейсів кінцевого користувача;
- компонентного застосування як сукупності компонентів.

Таким чином, проектування компонентної програмної архітектури розширює поняття архітектурного стилю «клієнт-сервер», де кожен компонент реалізує свою функціональність в рамках контейнера зі стандартизованим інтерфейсом [29]. Отже, програмну архітектуру можна відобразити з використання UML-діаграм з використанням об’єктно-орієнтованого підходу, а це означає можливість впровадження та застосування повторного використання. Але компонентне проектування не дозволяє повною мірою реалізувати процес як забезпечення якості ПЗ, так і процес управління якості на етапах життєвого циклу [30].

3.5. Багатокритеріальне оцінювання альтернативних архітектурних патернів

3.5.1. Постановка задачі оцінювання

Процес вибору архітектурного рішення із урахуванням показників якості можна відобразити у вигляді структурної схеми [31], приведеної на рис. 3.6. Будемо вважати, що:

– $K_i^1, i = \overline{1,m1}$ – критерії якості системи відповідно до стандарту ISO/IEC 25010;

– $K_i^2, i = \overline{1,m2}$ – критерії якості певної архітектури;

– $A_i, i = \overline{1,n}$ – альтернативи архітектурних рішень.

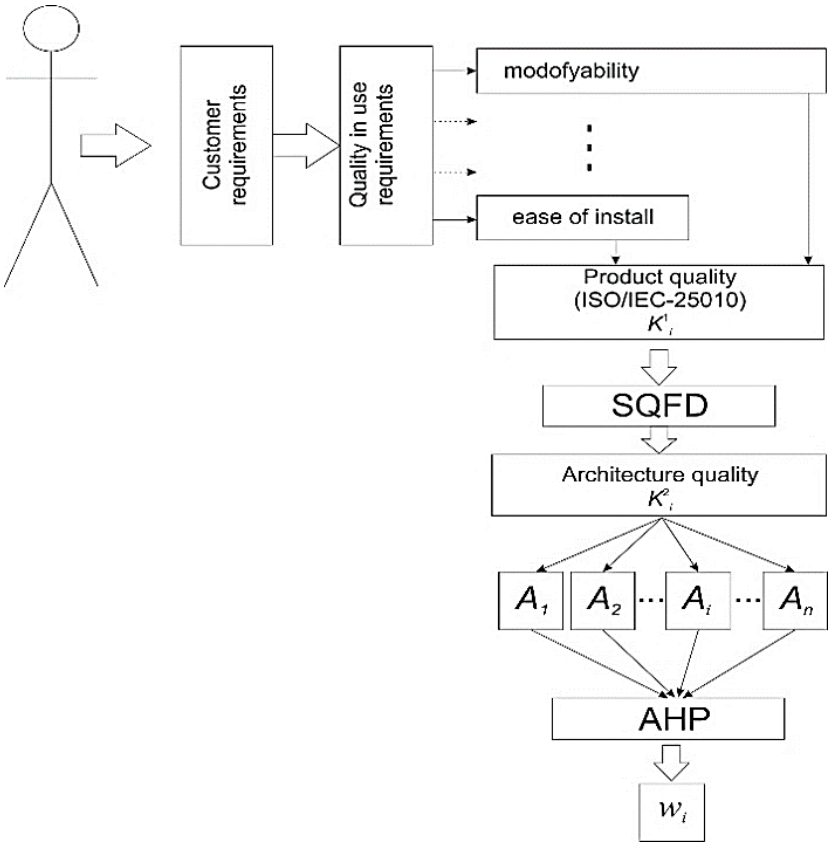

Рис. 3.6. Ієрархічне представлення задачі оптимізації архітектури

Сукупність критеріїв якості системі у використанні $\{K_i^1\}$ відповідно до стандарту ISO/IEC 25010 спільно із розробниками і замовником визначається і специфікується в модуль [31]. На основі даного переліку, що визначився в ході комунікації обох сторін, можна визначити критерії якості архітектури $\{K_i^2\}$.

За результатами проведеної задачі багатокритеріальної ієрархічної оптимізації, вирішеної найчастіше за використання методу аналізу ієрархій Сааті на основі сукупності критеріїв $\{K_i^1\}$ і $\{K_i^2\}$, згодом приймається архітектурне рішення [32].

Метод аналізу ієрархій (рис.3.7), що розглядається в даній роботі і використовується для рішення подібних задач, передбачає ваги для кожної альтернативи та критерію

w_i [31]. Ваги w_i знаходяться із заповнених експертами матриць попарних порівнянь B(b_{ij}), де b_{ij} - перевага *i*-тої альтернативи над *j*-ю і є узгодженими згідно $b_{ij} = w_i / w_i \ \ \forall b_{ij} \in B$. Вагові множники знаходяться як компоненти власного вектору матриці парних порівнянь, які відповідають максимальному характеристичному числу матриці [31].

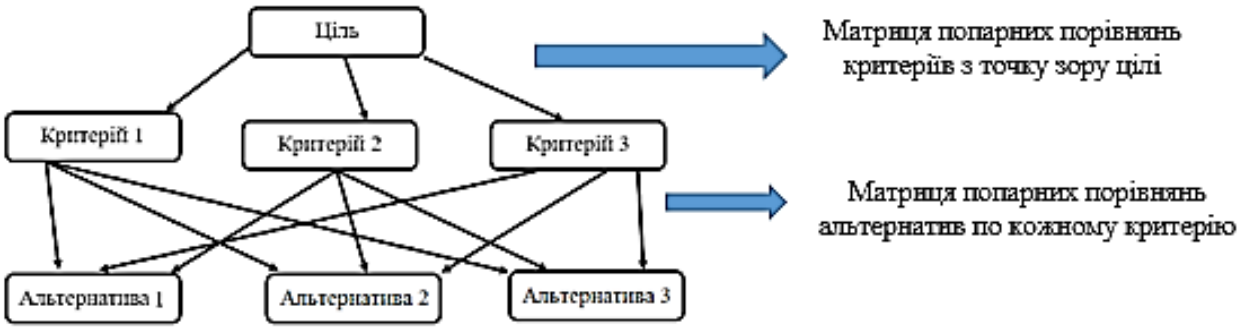

Рис. 3.7. Дерево критеріїв та альтернатив методу аналізу ієрархій.

Проблема може виникнути тоді, коли кількість альтернатив є досить значною, наприклад, n>9, матриця $B(b_{ij})$ не є узгодженою і її ранг буде відмінним від одиниці, тобто матриця буде мати декілька власних значень [32]. В такій ситуацій, з метою вирішення задачі вибору оптимальної архітектури можна використати алгоритм обчислення ваг ω_i з умови мінімізації неузгодженості попарної матриці порівнянь $B(b_{ij})$.

Нехай задано порогове значення узгодженості δ_t з метою можливості вибирати її ступінь [31]. Міру узгодженості запишемо у вигляді:

$$\left| \frac{w_i}{w_j} - b_{ij} \right| \le \delta_t \cdot b_{ij}, \quad \delta_t \ge 0, \tag{3.1}$$

Вагові множники w_i знаходяться наступним чином:

$$\min_{\{w_i\}} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(w_i - b_{ij} w_j\right)^2$$
$$a_i \le w_i \le c_i, \quad i = \overline{1,n} \tag{3.2}$$

$$-\delta_t \cdot b_{ij} \cdot w_j \le w_i - b_{ij} \cdot w_j \le \delta_t \cdot b_{ij} \cdot w_j; i, j = \overline{1,n}. \tag{3.3}$$

Зведемо (3.2), (3.3) нелінійного програмування до еквівалентної задачі лінійного програмування:

$$\min_{\{w_i\}} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(y_{ij}^{+} - y_{ij}^{-}\right)$$
$$w_i \ge a, i = \overline{1,n}, \tag{3.4}$$
$$w_i - b_{ij} w_j = y_{ij}^{+} - y_{ij}^{-}.$$

$$-\delta_t \cdot b_{ij} \cdot w_j \le w_i - b_{ij} \cdot w_j \le \delta_t \cdot b_{ij} \cdot w_j,$$
$$y_{ij}^{+}, y_{ij}^{-} \ge 0; i, j = \overline{1,n}. \tag{3.5}$$

Їхня еквівалентність слідує з того, що вектори обмежень при y_{ij}^{+} та y_{ij}^{+} є лінійно залежними і тому будь який розв'язок входить в y_{ij}^{+} або в y_{ij}^{+} і тому мінімум (3.4) відповідає мінімуму (3.2) [31].

Спочатку матриця попарного порівняння $B\left(b_{ij}^{s}\right)$ будується по кожному з критеріїв, де b_{ij}^{s} - перевага *i*-тої альтернативи над *j*-тою по реалізації *s*-го критерію. Потім визначаються набори вагових множників $\left\{w_i^s\right\}, i = \overline{1,s}, s = \overline{1,7}$ (як рішення задачі лінійного програмування) [31].

Далі треба здійснити ранжування за визначеною пріоритетністю альтернатив по множині критеріїв. Пріоритети критеріїв $\left\{K_i^2\right\}$ по їх впливу на $\left\{K_i^1\right\}$ заповнюються експертами у матрицю парних порівнянь $B^s\left(b_{ij}^{s}\right)$, де величина b_{ij}^{s} - переважання впливу K_i^2 над впливом критерію K_j^2 на K_s^1.

Таким чином, шляхом розв'язання (3.4), (3.5), отримується набір пріоритетів критеріїв якості архітектури $\left\{P_i^{1s}\right\}$, $i = \overline{1,m2}$, $s = \overline{1,m1}$ [31].

Вага альтернативи архітектури A_i відносно K_s^1 визначатиметься за формулою:

$$J_i^{1s} = \sum_{j=1}^{m2} p_j^{1s} \cdot w_i^j, \quad i = \overline{1,n}, \quad s = \overline{1,m1}, \tag{3.6}$$

де w_i^j –вагові множники, визначені на попередньому етапі.

На даному етапі можна проводити ранжування альтернатив $\{A_i\}$ за величиною $\{J_i^s\}$ для кожного $s = \overline{1,m1}$ в результаті чого ми отримаємо множини альтернатив і відповідні їм значення ваг $\{A_i^s, J_i^s\}$(s=1¯,m1) [31].

3.5.2. Вибір критеріальної функції

Критеріальна функція обирається з урахуванням особливостей самої задачі, схеми компромісів та принципу, яким керується особа, що приймає рішення. Приймання багакритеріальних рішень відбувається в області компромісів Парето, де покращення одних критеріїв може здійснюватися за рахунок погіршення інших, менш важливих [32]. Цю область (при випуклій множині значень критеріїв) можна визначити за допомогою:

$$\mathrm{X} = \bigcup_{x \in X_\alpha} \arg \min_{x \in X} \sum_{j=1}^{m} \alpha_j K_j(x), \tag{3.7}$$

де X_α – область визначення рішень, K_j – значення частинних критеріїв, $\alpha = \{\alpha_j\}_{j=1}^m$ – параметр, визначений на множині [31]:

$$X_\alpha = \left\{ \alpha \middle| \sum_{j=1}^{m} \alpha_j = 1, \ \alpha_j \geq 0 \right\}_{j=1}^{m}. \tag{3.8}$$

Рішення можна отримати з (3.7) на основі прийнятої схеми компромісів при певних значеннях α_i, визначених особою, що приймає рішення. Виходячи із (3.7) можна записати наступну критеріальну функцію $Q(A)=\sum_{j=1}^{m}\alpha_j\left(R_j-K_j(A)\right)$ для випадку мінімізації. Пронормуємо значеннями обмежень для можливості порівняння частинних критеріїв різної природи [31]:

$$Q(A)=\sum_{j=1}^{m}\alpha_j\left(1-\overline{K}_j(A)\right). \tag{3.9}$$

Модель вибору оптимальної архітектури може бути записана у наступному вигляді, виходячи з того, що перетворення є монотонним і за теоремою Гермейєра це не змінює результатів порівняння [31]:

$$A_{opt}=\arg\min_{A_i\in A}\sum_{j=1}^{m}\alpha_j\left(1-\overline{K}_j(A_i)\right),\ i=\overline{1,n}\ , \tag{3.10}$$

де А– множина альтернатив архітектури $A=\{A_i\},\ i=\overline{1,n}$.

Отримана функція (3.9) є лише лінійною апроксимацією в малому околі деяких базових точок, що є її недоліком, оскільки при розширенні області визначення застосування це може привести до значних помилок у процесі прийняття рішень [32]. Скориставшись принципом «подалі від обмежень», виберемо нелінійну критеріальну функцію:

$$Q(A_i)=\sum_{j=1}^{m}\alpha_j\left(1-\overline{K}_j(A_i)\right)^{-1},\ i=\overline{1,n}\ . \tag{3.11}$$

У разі наближення значень деяких критеріїв до граничних обмежень буде реалізуватись мінімаксна модель прийняття рішень: $A_{opt}=\arg\min_{A_i\in A}\max_{j=\overline{1,m}}K_j(A_i)$, а у ситуаціях, коли значення критеріїв далекі від обмежень, використовуватиметься модель інтеграційної оптимальності: $A_{opt}=\arg\min_{A_i\in A}\sum_{j=1}^{m}\alpha_j\left(1-\overline{K}_j(A_i)\right)^{-1}$ [31]. У випадку

наявності і мінімуючих, і максимуючих критеріїв, (3.11) матиме наступний вигляд: $Q(A_i)=\sum_{j\in L_1}\alpha_j\left(1-\overline{K}_j(A_i)\right)^{-1}+\sum_{j\in L_2}\alpha_j\left(\overline{K}_j(A_i)-1\right)^{-1}$, $i=\overline{1,n}$, де L_1 – множина індексів критеріїв, які мінімізуються, L_2 – індекси критеріїв, що максимізуються.

Таким чином, прийняти оптимальне архітектурне рішення можна визначивши α_j та підставши їх в (3.11). І з отриманих результатів вибрати кращу альтернативу. Отримане рішення аналізується і за необхідно робиться додаткове корегування α_j, так, щоб послідовність паретовських рішень сходилась до оптимуму [32].

ВИСНОВКИ ДО РОЗДІЛУ 3

Отже, кожен підрозділ глобальної розробки повинен керуватися і підтримуватися принаймні одним архітектором, який відповідає за розробку архітектури, дотримання її в ПЗ і забезпечує відповідність процесів архітектурним вимогам. Можна зробити висновок, що архітектори відіграють цілу потрійну роль у підрозділах.

З одного боку, вони приймають рішення щодо архітектури комплексу і створюють відповідні моделі, з іншого боку - забезпечують керівництво та підтримують свою бригаду у виконанні архітектурних стандартів. Майже у всіх мережах обміну інформацією архітектори формують центральні вузли в рамках міжкомандного та внутрішньо-командного координування інформації, при цьому віддається перевага саме прямому спілкуванню [27].

Зокрема, архітектори щоденно обмінюються інформацією зі своїми командами, а внутрішньо командний обмін між архітекторами та їх командами зазвичай характеризується мережею загального каналу зв’язку.

РОЗДІЛ 4

МЕТОДИ КООРДИНАЦІЇ КОМАНД ТА УПРАВЛІННЯ ЗНАННЯМИ ПРИ РОЗРОБЦІ ВЕЛИКИХ ПРОЕКТІВ В ГНУЧКИХ ТЕХНОЛОГІЯХ

З появою та приходом гнучких методів підхід до розробки ПЗ зазнав значних змін, що призвело до зовсім іншого способу планування та управління програмними проектами з посиленим акцентом на те, що розробка ПЗ є роботою спільною. З огляду на це, необхідно зробити наголос на гострій необхідності «арен» для планування, синхронізації та огляду. І хоча ці, так звані, «арени» від початку були розроблені для використання у невеликих самоврядних команд, на сьогоднішній день застосовуються і для великих програм розробки. Основною проблемою масштабних проектів є координація роботи багатьох команд [33].

Виникає це внаслідок того, що великі програми, як правило, включають технічну та організаційну складність, а це підпорядковує і велику кількість зацікавлених сторін, і учасників програми, і кількість вимог, рядків програмного коду та часто дуже складні взаємозалежності між завданнями та командами.

Отже, великомасштабні програми із використанням гнучких підходів у своїй роботі, дуже ризикують відсутністю взаємодії та труднощами у спілкуванні, оскільки більшість комунікацій здійснюються усно в командах. Така складність, як правило, негативно впливає на ефективність проекту і аби досягти успіху необхідно вміти керувати цією складністю [34]. Координація має вирішальне значення, оскільки робота виконується одночасно багатьма командами розробників, й нижче описано підходи координації для команд, що поєднують спритний та традиційний підходи.

4.1. Координаційні процеси в межах командної роботи

Під поняттям координації визначають управління діяльнісними залежностями, які включають у себе розподіл ресурсів, синхронізацію діяльності та інші необхідні заходи [33]. Під механізмами координації розуміють заходи з організації, які дають людям змогу діяти у межах колективної діяльності й включають у себе безпосередній контроль, взаємне коригування, стандартизацію роботи, результатів, навичок та норм [33]. Діяльність з безпосереднього контролю передбачає координаційні процеси та видачу директив на виконання певної роботи від однієї людини, яка є відповідальною за це, а взаємне коригування – процес пристосування працівників один до одного з ході спільної роботи. Інші механізми, що були зазначені при наданні визначень понять, включають в себе різні види стандартизації, такі як стандартизація роботи, продуктивності, навичок, знань, норм [34].

Як випливає із попереднього розділу, основними факторами необхідності міжкомандної координації є:

- *невизначеність завдань*, що спонукається через складність роботи, що виконується організаційним підрозділом, та її варіативність;
- *взаємозалежність завдань*, яку можна розглядати як міру залежності організаційного підрозділу від інших для виконання своєї роботи.
- *розмір робочої одиниці*, яка визначає кількість людей у робочій одиниці.

На сьогоднішній день визначають три режими координації – груповий, індивідуальний та безособовий (табл. 4.1) [33].

Таблиця 4.1.

Режими координації, їхній опис та основні механізми

Режими особистої координації	Опис режиму	Механізми
Груповий	Механізми координації, які передбачають взаємне коригування через зустрічі у кількості більше двох працівників (групи).	Механізми запланованих та незапланованих зустрічей.
Індивідуальний	Механізми координації із взаємного коригування через вертикальну або горизонтальну комунікацію між окремими ролями працівників.	Механізми горизонтальних та вертикальних каналів.
Безособовий	Механізм координації безособового кодифікованого критерію дій.	Механізм «креслення» дій.

Розглянемо більш детально кожен із них.

Груповий режим особистої координації – механізм взаємного коригування, що поширюється на групу рольових осіб шляхом застосування запланованих або позапланових зустрічей, де приймає участь більше двох учасників.

Заплановані зустрічі – це зустрічі, що використовуються для запланованого спілкування і можуть являти собою демонстраційні збори з визначеною частотою, керівницькі зустрічі Metascrum, зустрічі для визначення залежностей у завданнях перед призначенням роботи командам, зустрічі Scrum of Scrums, щоденні засідання, ретроспективи, обідні семінари тощо [33].

Позапланові зустрічі – зустрічі, що використовуються для спілкування поза планом і проводяться частіш за все через відкритий робочий простір. Можуть являти собою обговорення рішень біля дошок, малювання та створення ескізів. На дошках можуть бути розміщені управлінські таблиці проектів для перегляду статусу та прогресу команд [33].

Також, можуть бути застосовані, так званні віртуальні зустрічі (груповий чат), де члени команди можуть сповістити про щось, попросити допомоги та надати поради. При цьому, проглядається залежність, що керівники проектів, які використовують підхід із груповою взаємодією у запланованих зустрічах, можуть досягати більшого контролю над роботою у невизначених ситуаціях [34].

Індивідуальний режим особистої координації - механізм взаємного коригування задач через вертикальні і/або горизонтальні зв'язні канали рольовими особами. Горизонтальний канал приймає функцію зв'язування через окремого члена підрозділу, який безпосередньо спілкується з іншими учасниками за принципом «тет-а-тет» поза ієрархічними відносинами, в той час, як механізми вертикальної комунікації, як правило, є лініями між керівниками та керівниками підрозділів.

Цей режим у великій кількості випадків є одним із вирішальних під час розв'язування взаємозалежностей завдань та збереження графіку, його ще можна охарактеризувати як прямий контакт між експертами [33]. Гарною та продуктивною є також практика широкої особистісної координації робітників, що працюють над єдиними цілями та проектами. Обіди, кава-брейки та інші подібні заходи є важливими соціальними «аренами» для реалізації координаційних механізмів в ході проекту.

Безособовий режим особистої координації – запрограмований або кодифікований механізм координації, що вимагає мінімального вербального людського спілкування після реалізації. Основними практичними механізмами, при цьому, є програмний план (електронні таблиці, трекери, дошки із наліпками), керівні принципи та контрольні списки [33]. Тобто, трекер проблем

у будь-якій його реалізації, представляє собою програмний план та є важливим координаційним механізмом на рівні проекту, аналогічно фізична дошка – на рівні команди.

Існує також така реалізація безособового режиму, як вікі – веб-сайт із обмеженим доступом, де зберігаються та піддаються колективним модифікаціям усі документи, інструкції та контрольні списки з описом процесів.

Отже, визначені механізми координації у розглянутих режимах можна згуртувати у вигляді таблиці 4.2 [33].

Таблиця 4.2.

Практики механізмів координації

Режими координації	Механізми координації	Опис механізму
Груповий	Проектна зустріч з архітектури	Зустріч між архітекторами команд (провідними та технічними)
	Настільні дискусії	Зустріч навколо дошки, що показує завдання команди та їхній прогрес
	Зустріч бізнес-проекту	Зустріч власників продукту та архітекторів команд
	Демо-зустріч	Зустріч-показ реалізації зацікавленим сторонам
	«Досвід форуму»	Зустріч для обмін досвіду між командами з технічних питань
	«Обідні семінари»	Зустріч для обміну досвідом між групами з самостійних тем
	Meta-Scrum	Формальна зустріч керівників проекту щодо прогресу
	Відкриті простори	Зустріч неформального формату на теми, підняті учасниками
	Опен спейс	Команди розташовані на одному

		поверсі у відкритому просторі

Продовження таблиці 4.2

	Ретроспектива	Зустріч в кінці спринту для обговорення змін
	Scrum-of-Scrums	Зустріч представників проектних команд щодо прогресу
	Технічний куточок	Позапланова проектна зустріч для обміну досвідом з технічних питань
	Засідання тест-проекту	Зустріч з провідним тестувальником і усією командою QA
Індивідуальний	Ротація одиниць команди	Ротація членів між командами в рамках проектів
	Замовник на місці	Представники замовника сидять у відкритому офісному приміщенні для надання консультацій
	Пряме спілкування	Легкий доступ до людей з інших команд у опен спейсі
Безособовий	Миттєві повідомлення	Інструмент, що сприяє асинхронному спілкуванню всіх учасників проекту (чати)
	Master-plan	Епос, план основних функцій, які будуть включені в реалізацію
	Рекомендації з архітектури	Опис технічних рішень проекту і стандартів розробки
	Набори команд	Описи з очікуваннями для роботи команд та між командами на порталі вікі
	Опис рішень	Опис реалізацій, включаючи детальні історії користувачів

Перші дослідження координаційних явищ мали висновки, що координація – статичний процес, але пізніше це було піддано критиці, оскільки фактори, що впливають на режими координації можуть змінюватися в ході роботи над проектом – невизначеність завдань, взаємозалежність завдань та розмір робочої одиниці. Наприклад, невизначеність завдань може змінитися в ту чи іншу сторону через розширення галузевих знань та технічних рішень учасниками проекту, внаслідок залучення нових осіб; ступінь взаємозалежності завдань – внаслідок прийняття якось технічного рішення під час виконання проекту чи також через залучення нових робітників [34]. Зростання взаємозалежності команди також доволі поширене явище, особливо в індивідуальному режимі (наприклад, «за кавою» або біля дошки). Однак, також є поширеним протилежне явище, а саме – зростання використання безособової та групової координації, що може бути викликане розширенням кадрів підрозділу. Що ще раз доводить мінливість координаційних процесів, тому що механізми координації адаптуються до невизначеності, новизни чи змін з плином часу.

На сьогоднішній день великомасштабний розвиток не втрачає своїх тенденцій та продовжує поширюватись світом, набуваючи все більш широкого інтересу. І не зважаючи на те, що вже існує ціла існує низка методологій та різноманітних консультації щодо глобальної розробки, але це питання все ще перебуває у зародковому стані, на жаль. Однією з головних проблем є координація роботи багатьох команд і висновки в галузі координації наголошують, що крос-командні процеси є більш важливими і складними, ніж процеси в межах команди для виконання багатопоточних систем [33].

4.2. Емпіричні дослідження процесів координації команд

Як вже зазначалось, досягнення ефективної міжкомандної координації - одна з найактуальніших проблем масштабної розробки ПЗ. Гібридні підходи традиційної та гнучкої розробки обіцяють поєднувати довгострокове планування на міжкомандному рівні (МКР) з гнучкістю та пристосованістю гнучкого розвитку на

командному рівні (КР), й попередні дослідження дійсно дозволяють припустити, що великі та складні проекти розвитку можуть отримати вигоду від поєднання гнучкої, органічної координації, властивої спритній колективній роботі, та структурованої, керованої планами координації традиційного управління проектами в гібридні форми [34]. Питання постає про те, чи справді можливий такий синтез та як гібридні підходи впливають на координацію, оскільки неадекватна реакція на взаємозалежність та неефективна координація є основними джерелами невдач проекту. Отже, необхідно зрозуміти появу та основні причини неефективної координації в гібридах традиційного та спритного підходів.

Велика кількість команд, що є взаємозалежними і працюють над спільною метою, представляють собою так звану структуру команд, яку в літературі ще називають багатоканальною системою (БКС) [35]. У такій БКС окремі команди мають як індивідуальні цілі, так і спільну мету з іншими командами, які, у свою чергу, мають залежність від вхідних даних, процесів та результатів принаймні з однією іншою командою. І в цьому випадку, для управління залежностями як всередині, так і між командами, головне значення має координація між групами.

Введемо поняття ефективності координації, під яким будемо розуміти стан координації, коли всі учасники програми розробки ПЗ БКС широко розуміють загальні цілі та пріоритети, що відбувається і коли, що їм, як команді, потрібно робити [35].

Отже, зосередимося на центральних механізмах, що перешкоджають ефективній координації в гібридних умовах. Існує досить традиційна стратегія для процесу планування та координації роботи більше десятка команд, а саме - створення центральної команди (ЦК), яка представляє собою команду, до складу якої можуть входити головний власник продукту (ГВП), архітектори та експерти з продуктів і т.д.

На рис. 4.1 зображено приклад налаштування БКС за такої стратегії. З самого верху розташовано ЦК, яка є відповідальною за планувальні процеси продуктів на високому рівні та координаційні процеси між групами розробників. Вона збирається з певною частотою і обговорює актуальні теми та плани на майбутні спринти; у той

же час – окремі команди (зображені на рис.4.1 на другому рівні) працюють у визначених ітераціях на основі спринту і складаються з уповноважених осіб. Таким чином, представлено широко застосовувану форму традиційного планування та координації на вищому рівні через ролі та ієрархію [35].

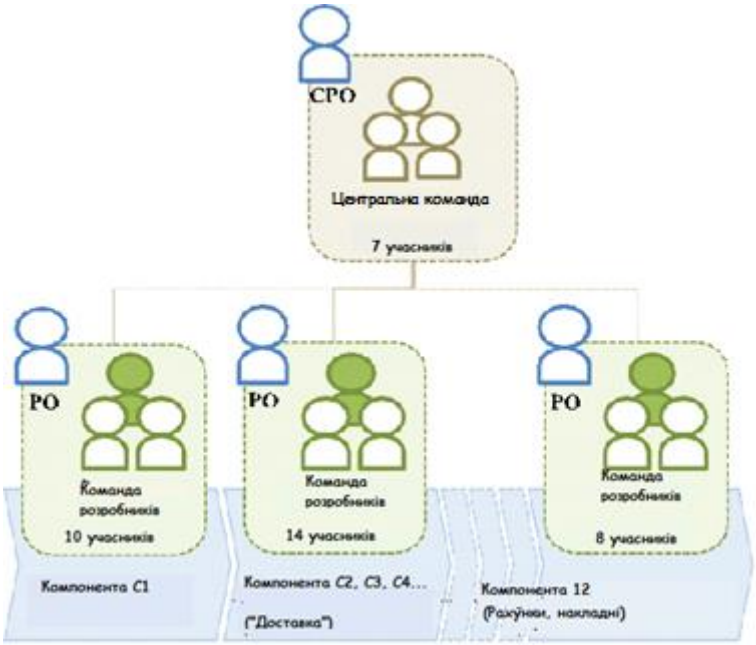

Рис. 4.1. Налаштування багатоканальної системи

Загалом виділяють два центральних елементи гібридного підходу:

- планування зверху вниз за допомогою ЦК;
- гнучкі методи та практики розробки ПЗ на рівні команди.

Більш докладно, ЦК, наприклад, створює курс роботи високого рівня у формі епосів та розподіляє їх серед окремих команд розробників. Епос описує вимоги високого рівня і поступово розбивається на більш детальні історії користувачів, а згодом - і на технічні завдання [35] (рис. 4.2).

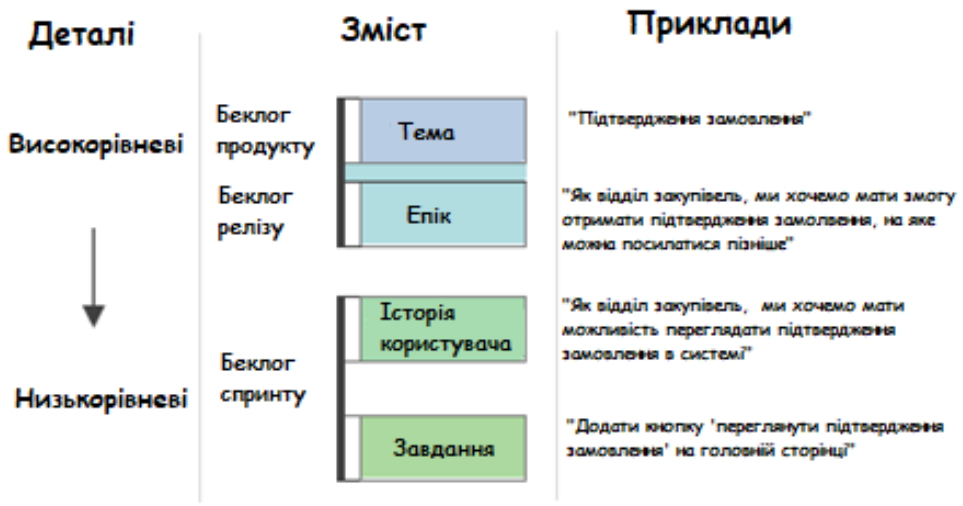

Рис. 4.2 Беклог ієрархія

За використання такого, здавалося б, традиційного та усталеного підходу команди стабільно страждають від того, що не можуть виявити та вирішити залежності від роботи інших команд. І дослідження стверджують, що більш високі рівні робочих залежностей збільшують схильність до подібних викликів проектів розробки ПЗ, а пояснюється це нездатністю людей виявляти неявні взаємозалежності робочого процесу та відстежувати зміну вимог до координації з часом [34]. Це може призводити до того, що команди «блокують» один одного, до проектних затримок та високого рівня загального невдоволення в підрозділі розробки.

Можна сказати, що в більшості випадків, коли проблеми координації стали очевидними, причиною може бути відсутність обізнаності про залежність. Невідомі залежності та охоплені потенційні конфлікти призводять до неефективної координації між групами розробників, що, в свою чергу, впливає на продуктивність та здатність команд до досягнення результатів [35]. Тобто, усвідомлення залежності є необхідним, хоча і недостатнім, попередником ефективної координації.

Також за використання ЦК планувальні заходи часто проводяться ще до фактичної реалізації вимог, це можуть буть такі процедури, як конкретизація, встановлення пріоритетів, оцінка, розподіл і т.д. За використання такої схеми ці процедури зміщуються на МКР та КР підрозділу розвитку, що викликає серйозні проблеми пізніше під час впровадження [35]. Компетентність третього виміру (див.

рис. 4.2) відноситься до конкретних технічних знань, професійного досвіду кадрів та досвіду проекту, що дозволяє певній ролі або організації, відповідальній за планування, виконувати свою роботу. Включаючи вищезазначені виміри збігу, можна визначити зсув планування як «несумісну схему фокусу, ступеня деталізації та компетентності у плануванні діяльності на КР стосовно МКР [35].

Таким чином, рис. 4.3 демонструє появу двох категорій: неправильність розподілу заходів з планування та відсутність обізнаності про залежність від відповідних основних концепцій та кодексів.

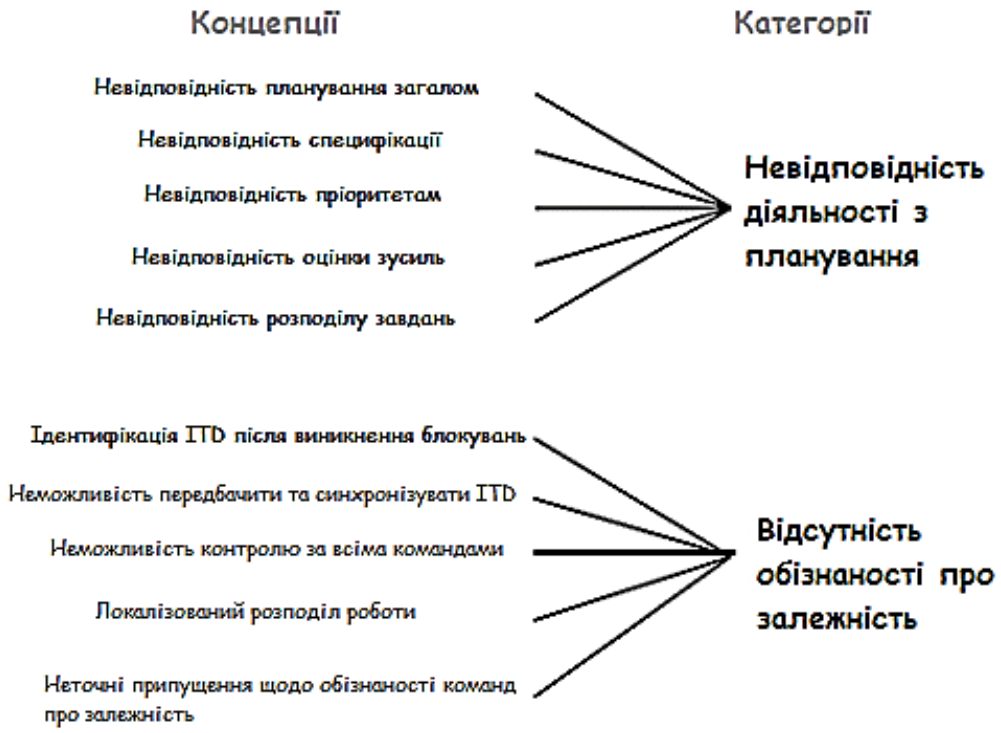

Рис. 4.3. Категорії основних складнощів виникаючих при координації

Прямим практичним наслідком цих висновків є те, що для покращення обізнаності про залежність та, в свою чергу, сприяння координації на МКР, можуть знадобитися регулярні зустрічі між командами (такі як планування спринту, огляди та ретроспективи, що охоплюють гнучкі принципи ітеративної доставки та співпраці, можуть відображатись на взаємодії командного рівня для полегшення узгодження планування). Це повинно створити двосторонній канал зворотного

зв'язку між окремими командами, з одного боку, та ЦК, з іншого, а також платформу для всіх команд, щоб усвідомити існуючі та потенційні залежності між ними [34].

В свою чергу, проблеми обізнаності можуть виникати внаслідок суперечливої орієнтації і погляду на зміст та часові рамки на МКР та КР, такі конфлікти можуть бути частково вирішені шляхом підготовки ітеративних планувальних заходів на МКР, розповсюдження результатів на МКР та надання можливості ітеративного процесу коригування між обома рівнями [35]. Таким чином, специфікація та встановлення пріоритетів на МКР можуть мати місце за умови наявності достатнього часу до початку наступного спринту команд розробників. Але завдяки цьому команди розробників зможуть детально визначати пріоритетні завдання та передбачати виникаючі залежностей на МКР з їхнім подальшим вирішенням.

Розуміючи ці шкідливі механізми, результати було узагальнено в таблиці 4.3.

Таблиця 4.3

Вплив невідповідності планування між МКР та КР на рівень обізнаності про залежність

ДІЯЛЬНІСТЬ З ПЛАНУВАННЯ	**Міжкомандний рівень (МКР)**	**Командний рівень (КР)**
СПЕЦИФІКАЦІЯ	Попередня специфікація високорівневих епіків, завдяки чому стане цілісна картина функцій стане доступною	Вчасна специфікація найважливіших історій користувачів для поточного спринту, але з високими технічними деталями
	Причин невідповідностей специфікації - центральна команда намагається заздалегідь вказати міжгрупові залежності, але їй не вистачає технічних деталей для виявлення всіх; - команди розробників мають технічні досвід для виявлення взаємозалежностей команд для своїх завдань, але їм бракує часу та «картинки» високого рівня, щоб вирішити їх з іншими командами.	
	Наслідки невідповідності специфікації - виявлені МКР залежності відомі єдиній команді розробників, отже, інші команди, які страждають, не можуть врахувати відсутність їхньої обізнаності про залежності.	

Продовження таблиці 4.3

ПРІОРИТЕЗАЦІЯ	Центральна команда проводить абсолютну, високорівневу пріоритезацію з фокусуванням на потреби ринку та загальний план випуску.	Команда власника продукту ранкує елементи беклогу для наступних спринтів, базуючись на основі технічних знань та фактичного прогресу, про який їм повідомляє їх команда розробників.
	Причини невідповідності пріоритетів - центральна команда встановлює багато, однаково високих абсолютних пріоритетів для необхідних функцій і рідко визначає перепріоритети; - команди розробників вирішують внутрішні відставання за допомогою локальних пріоритетів КР без інших команд.	
	Наслідки невідповідності пріоритетів - пріоритезація КР усуває посилені залежності від відставання посилених залежностей МКР; - груба пріоритезація МКР і непрозорі, потенційно суперечливі пріоритети КР →відсутність обізнаності про залежність	
ЕСТИМАЦІЯ	Центральна команда наближено апроксимує розмір теми на основі досвіду, при цьому плануючи відставання від термінів випуску.	Команди власника продукту та розробників спільно оцінюють точні зусилля щодо відкритих завдань наступного спринту шляхом внутрішніх обговорень.
	Причини невідповідності оцінки - план випуску, заснований на неточних оцінках центральної команди є основою для розподілу історій користувачів між командами; - команда розробників коригує оцінки поетапно за спринт, блокує надмірні зусилля (хоча інші команда можуть залежати від цього) і повідомляють змінені оцінки лише після спринту.	
	Наслідки невідповідності оцінки - непрозорі оцінки КР → незрозумілі дати передачі даних від інших команд→ посилені МКР залежності; - неточна оцінка зусиль зверху вниз та відсутність механізмів зворотного зв'язку для коригування оцінки знизу вгору → відсутність усвідомлення залежності.	
РОЗПОДІЛ	Центральна команда виділяє грубо теми для спеціалізованих команд компонентів на основі огляду архітектури та досвіду продукції.	Команда розробників самостійно розподіляє завдання щодня/спринтом під час щоденних зустрічей на основі можливостей та досвіду людей.

Продовження таблиці 4.3

РОЗПОДІЛ	**Причини невідповідності розподілу** - розподіл тем зверху вниз, призначений для відповідності досвіду команди та запобігання залежностям МКР, але відбувається незалежно від детальних пріоритетів та оцінок КР; - команди розробників знають про загальні обов'язки інших команд, але мало знають про їхні призначені теми в даний час через індивідуальні сеанси передачі між центральною командою та командою власника продукту.
	Наслідки невідповідності розподілу - розподіл тем заздалегідь без детальної специфікації, встановлення пріоритетів та оцінки, запобігає ідентифікації залежності від МКР → відсутність обізнаності про залежність; - залежності МКР виявляють лише команди розробників під час спринту, але командам не вистачає огляду та шансів на спеціальне регулювання/перерозподіл.

Виходячи із розробленої моделі процесу (табл. 4.3), можна зробити наступні висновки щодо процесів специфікації, пріоритезації, естимації та розподілу.

Специфікація

По-перше, у ЦК відсутній детальний технічний огляд, щоб зробити специфікацію, яка має на меті попереджати всілякі потенційні залежності, ідентифікуючи їх на ранніх стадіях [35].

Отже, для традиційної специфікації модульних завдань зверху вниз передбачений ступінь загрози є просто занадто низьким. А оскільки ЦК, як правило, не враховує відгуки команди розробників щодо детальних специфікацій, то вони не компенсують цей недолік, і самим командам треба шукати інші команди, які постраждали від нещодавно виявлених взаємозалежностей.

По-друге, команди мають технічний досвід для виявлення взаємозв’язків між командами для вказаних історій користувачів безпосередньо перед наступним спринтом, але їм може не вистачати часу на виконання та картини високого рівня, щоб вирішити їх з іншими командами [35].

Отже, виявлені залежності на МКР можуть бути відомі лише окремим командам розробників, а інші (в майбутньому постраждалі) команди не можуть їх врахувати. Так, розбіжність специфікації (через недостатність деталізації

специфікації завдання зверху вниз) та відсутності взаємодії для розробки завдання знизу призводить до відсутності усвідомлення залежності.

Розстановка пріоритетів

Знову ж таки, дві проблеми через неузгодженість підходів на МКР на КР. По-перше, ЦК часто встановлює велику кількість високих абсолютних пріоритетів для обов'язкових функцій з точки зору бізнесу і, як наслідок, командам важко визначити пріоритет роботи, що залишилася від попередніх спринтів, порівняно з новими епосами [35].

По-друге, завдяки своїм внутрішнім пріоритетам, команди вирішують внутрішнє відставання [35]. Однак, оскільки команди розробників не знають про можливий перерозподіл пріоритетів та послідовність робочих кроків у інших командах, це часто посилює наслідки існуючих міжгрупових залежностей.

Отже, можна стверджувати, що груба розстановка пріоритетів на МКР, разом із непрозорими, потенційно конфліктними перерозподілами на рівні команди, запобігають усвідомленню залежностей.

Естимація

По-перше, оцінки ЦК часто є дуже неточними, і проблема в тому, що, все ж таки, вони складають основу для розповсюдження епосів серед команд розробників. Оцінки на МКР на КР також розходяться, оскільки набагато точніші оцінки, зроблені окремими командами, зберігаються внутрішньо, і не можуть бути враховані іншими командами. Якщо відсутній стандартизований процес, протягом якого команди могли б передавати свої спільно уточнені оцінки зусиль, жодного узгодження оцінки не може бути здійснено.

По-друге, непрозорі оцінки рівня команди також можуть заважати командам розробників точно знати, коли очікувати передачі від інших команд. Невідповідність оцінки може бути узагальнена як неточна оцінка зусиль зверху вниз та відсутність механізмів зворотного зв'язку для коригування оцінки знизу вгору і разом це заважає усвідомленню залежності.

Розподіл

Після уточнення, встановлення пріоритетів та оцінки задач, останнім кроком у процесі планування є їхній розподіл для реалізації командам. На МКР ЦК розподіляє епоси для спеціалізованих команд і, очевидно, що такий попередній розподіл без детальної специфікації, встановлення пріоритетів та оцінки перешкоджає виявленню та передачі залежності. І це знову призводить до відсутності усвідомлення залежності.

Таким чином, обізнаність про залежність необхідна для ефективної координації всіх команд, однак її наявність аж ніяк не гарантує ефективної координації. Це змушує дійти висновку, що наступні заходи є необхідними, хоча і недостатніми для усвідомлення залежності [35]:

1) проінформованість про залежності як МКР, так і на КР;

2) вирівнювання планування всіх фаз (специфікації, пріоритетизації, естимації та розподілу).

Згідно зазначених вище пропозицій 1) і 2), можна запропонувати модель процесу ефективної координації в гібридних умовах (рис. 4.4):

Рис. 4.4. Взаємозв'язок між вирівнюванням діяльності з планування, обізнаністю про залежність та ефективністю координації

Отримана модель процесу ефективної координації в гібридних традиційно-спритних умовах розвитку описує необхідні події, які відбуваються як передумови ефективної координації. Звісно, це не вичерпне пояснення ефективної координації, а модель, яка передбачає, що координація може стати ефективною лише тоді, коли існує виконання та дотримання вказаних передумов, і це не означає, «чим більш узгодженими планувальними заходами, тим вища обізнаність про залежність та вища ефективність координації» [34]. Натомість, вона визначає послідовність

необхідних передумов, які мають відбутися до того, як стане можливою ефективна координація. Таким чином, модель стверджує, що коли планування діяльності не узгоджується, це перешкоджає усвідомленню залежності всіх суб'єктів; а коли немає загальної обізнаності про залежність - координація стає неефективною.

Отже, невідповідність планувальних заходів постійно спричиняє відсутність обізнаності про залежність, що, як правило, призводить до неефективної координації [35]. Тобто, ефективна координація неможлива без усвідомлення залежності, а це, у свою чергу, неможливе без узгоджених заходів з планування МКР та КР. Звісно, низка інших причин також може призвести до неефективної координації, але недостатню обізнаність про залежність можна чітко визначити як домінуючу проблему для неефективної координації.

4.3. Математична модель управління процесами координації

Організація процесу управління в системах, що слабо формалізуються (СФ), все більш активно і частіше входить в коло проблем, що вирішуються за допомогою сучасної теорії управління [36]. Взагалі, поняття слабкої формалізації пов'язують із запропонованою Гербертом Саймоном класифікацією управлінських проблем і прийняття рішень на наступні типи [37]:

- добре структуровані, або кількісно сформульовані;
- неструктуровані, або якісно виражені;
- слабо/погано структуровані, або змішані.

У літературі є різні визначення поняття слабкої формалізації стосовно об'єктів управління, проте в цілому, процесом СФ називається динамічний процес, що відноситься до класу неструктурованих і слабо структурованих проблем управління [38]. При цьому СФ системи можна охарактеризувати наступним чином:

- унікальність процесів системи;
- неоднорідність шкал виміру параметрів;
- нелінійний характер взаємозв'язку параметрів;

– багаторівнева ієрархічна організація взаємозалежності процесів;

– їх структура і/або функціонування не можуть бути описані формальними моделями у вигляді деякої сукупності рівнянь;

– вони є активними і мають «свободу волі»;

– цілі їх існування і критерії управління ними також не можуть бути формалізовані і змінюються в часі.

Традиційно до СФ, в першу чергу, відносять об'єкти управління національних органів класифікації - соціально-економічні та організаційно-технічні системи, у технічних додатках до СФ систем, як правило, відносять великомасштабні виробничі об'єкти, що розглядаються як великі системи, моделювання яких ускладнено внаслідок їх розмірності [39].

СФ характер таких технологічних процесів доцільно моделювати за допомогою СФ критеріїв, а весь процес - базуючись на принципах самоорганізації і координаційного управління [40]. Розглянемо можливості організації координаційного управління такими процесами з безпосереднім включенням людини як носія інтегрального критерію в контур управління.

Основні ідеї координації пояснюються на рис. 4.5. Нехай процес моделюється системою S, що складається з (n + 2) підсистем: вищої управляючої системи (координатора, або ЦК) C_0, n нижчих управляючих систем C_1, ..., C_n і керованого процесу P. Процес P складається з окремих підпроцесів P_i, взаємодія між якими здійснюється за допомогою сполучних сигналів u_i.

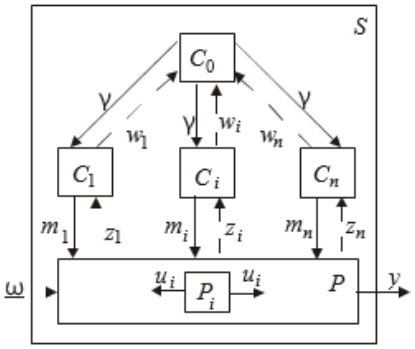

Рис. 4.5. Дворівнева ієрархічна система

Жоден з елементів системи S не вирішуєсамостійно її глобальне завдання D_S, хоча останнє і визначене в термінах всього процесу. Кожна з підсистем вирішує свої завдання: D_0 відповідає меті координатора C_0, $D(\gamma) = \{D_1(\gamma), ..., D_n(\gamma)\}$ - цілям нижчестоящих управляючих систем $C_1,..,C_n$.

У загальному випадку всі ці цілі D_i є різними, проте цілі елементів нижчого рівня залежать від координуючого параметра γ, одержуваного від координатора C_0. Координатор C_0, в свою чергу, обирає значення γ так, щоб забезпечити рішення свого власного завдання. За умови наявності між D_S, D_0 і $D(\gamma)$ певних закономірностей, глобальна мета системи S може бути досягнута. Для створення цих закономірностей координатор C_0 може використовувати різні шляхи параметризації: вимірюючи або множину цільових функцій (ЦФ) підсистем C_1, ..., C_n, або множину сполучних входів для керованих ними підпроцесів, або те й інше разом.

Процеси з СФ критеріями, як правило, мають цілком певну і визначену матрицю технологічних зв'язків, тому множина $U \supset u_i$ є недоступною для управління, крім елемента u_n, який одночасно є глобальним вихідним сигналом процесу в цілому, і в цьому випадку може бути застосована параметризація шляхом прогнозування взаємозв'язків [40]: управляючі підсистеми C_i, $1 \geq i \geq n$, отримують від координатора C_0 не тільки значення параметра β своєї цільової функції, а й прогнозні значення сполучних сигналів α, і включають їх в рішення своїх завдань $D(\gamma)$, тобто $\gamma = \{\alpha, \beta\}$. Тоді рішення задачі координації в змістовному сенсі зводиться до підбору поточних значень β, які при дотриманні матриці технологічних зв'язків, тобто при збігу прогнозних і реальних значень α, доставляють максимум глобальної функції якості системи в цілому.

Математична модель процесу (рис. 4.5) будується на рівні теоретико-множинного опису системи S [38] в каузальній

$$\boldsymbol{S}: \boldsymbol{X} \rightarrow \boldsymbol{Y} \qquad (4.1)$$

та критеріальній

$$\boldsymbol{G}: \boldsymbol{X} \times \boldsymbol{Y} \rightarrow \boldsymbol{V} \tag{4.2}$$

формах.

У вхідному об'єкті X процесу P є за доцільне виділити:

- множину M управляючих сигналів m, m ⊂ M;
- множину Ω зовнішніх збурень ω, ω ⊂ Ω.

Тобто, можна вважати, що процес P реалізує наступне відображення:

$$\boldsymbol{P}: \boldsymbol{M} \times \boldsymbol{\Omega} \rightarrow \boldsymbol{Y}\,. \tag{4.3}$$

Оціночний об'єкт V системи S можна інтерпретувати як множину оцінок якості кінцевого продукту процесу, що розглядається [38]. Передбачається, що множину M представлено у вигляді декартового добутку n підмножин локальних управляючих сигналів M_i:

$$\mathrm{M} = \mathrm{M}_{\boldsymbol{1}} \times \mathrm{M}_{\boldsymbol{2}} \times \ldots \times \mathrm{M}_{\boldsymbol{n}}, \boldsymbol{1} < \boldsymbol{i} < \boldsymbol{n}. \tag{4.4}$$

І кожен локальний управляючий сигнал $m_i \subset M$ формується локальною управляючою системою Ci і впливає на підпроцеси P_i. Тоді управляючий сигнал для процесу в цілому може бути записаний у вигляді вектора

$$\boldsymbol{m} = \{\boldsymbol{m_i}\}. \tag{4.5}$$

Координатор C_0 впливає на локальні управляючі системи C_i за допомогою координуючих сигналів

$$\boldsymbol{\gamma} = (\boldsymbol{\gamma_1}, \ldots, \boldsymbol{\gamma_n}), \tag{4.6}$$

які утворюють множину Г, $\gamma \subset \Gamma$, причому на вхід і-ої локальної управляючої системи надходить тільки компонента γ_i.

Опишемо співвідношення між процесом Р в цілому і його підпроцесами P_i, важаючи, що $U = U_1 \times U_2 \times \ldots \times U_n$. Визначимо функції Н на множині М×Y і $\bar{P}$ на множині М×U×Ω у вигляді

$$\boldsymbol{H(m,y) = (H_1(m,y), \ldots, H_n(m,y))}; \quad (4.7)$$

$$\boldsymbol{\bar{P}\,(m,u,\omega) = (P_1(m,u,\omega), \ldots, P_n\,(m,u,\omega))}. \quad (4.8)$$

Підпроцеси пов'язані між собою, якщо умова

$$\boldsymbol{y = \bar{P}(m, H(m,y), \omega) \Leftrightarrow y = \bar{P}(m,\omega)} \quad (4.9)$$

виконується для всіх (*m*, *y*, ω) ⊂ *M*×*Y*×Ω. Із (4.9) випливає, що *u* є результатом відображення

$$\boldsymbol{K: M \times \Omega \rightarrow U,} \quad (4.10)$$

або, у покомпонентному записі [39],

$$\boldsymbol{K_i: M \times \Omega \rightarrow U_i,} \quad (4.11)$$

яке, у свою чергу, визначається рівнянням

$$\boldsymbol{K\,(m,\omega) = H\,(m, P(m,\omega))}. \quad (4.12)$$

Таким чином, функція К - функція взаємодії підпроцесів [38] - визначається із співвідношення

$$\boldsymbol{P(m,\omega) = \bar{P}(m, K(m,\omega), \omega)}. \quad (4.13)$$

При стаціонарності системи S і навколишнього середовища, множина Ω може бути звужена до одного елемента [39]. Припускаючи, що система в цілому працює в умовах визначеності, можна записати

$$\boldsymbol{P}: \boldsymbol{M} \times \boldsymbol{U} \rightarrow \boldsymbol{Y}; \tag{4.14}$$

$$\boldsymbol{G}: \boldsymbol{M} \times \boldsymbol{U} \rightarrow \boldsymbol{V}; \tag{4.15}$$

$$\boldsymbol{P_i}: \boldsymbol{M} \times \boldsymbol{U_i} \rightarrow \boldsymbol{Y_i}\,. \tag{4.16}$$

Цільові властивості системи S характеризуються такими функціями:

– глобальна цільова функція $G: M \rightarrow V$ або

$$\boldsymbol{g}(\boldsymbol{M}) = \boldsymbol{G}(\boldsymbol{m}, \boldsymbol{P}(\boldsymbol{m})); \tag{4.17}$$

– міжрівнева функція якості Ψ_γ: $Vn \rightarrow V$ або

$$\boldsymbol{\Psi\gamma} = \{(\boldsymbol{g_{1\gamma}}(\boldsymbol{m_1}, \boldsymbol{K_1}(\boldsymbol{m})), \ldots, \boldsymbol{g_{n\gamma}}(\boldsymbol{m_n}, \boldsymbol{K_n}(\boldsymbol{m}))), \boldsymbol{g}(\boldsymbol{m}): \boldsymbol{m} \subseteq \boldsymbol{M}\}; \tag{4.18}$$

– глобальна цільова функція, що здається g_γ: $\Gamma \times M \times U \rightarrow V$ або

$$\boldsymbol{g_\gamma}(\boldsymbol{\gamma}, \boldsymbol{m}, \boldsymbol{u}) = \boldsymbol{\Psi\gamma}\,(\boldsymbol{g_{1\gamma}}\,(\boldsymbol{m_1}, \boldsymbol{u_1}), \ldots, \boldsymbol{g_{n\gamma}}(\boldsymbol{m_n}, \boldsymbol{u_n})); \tag{4.19}$$

істотно, що для будь-яких $\gamma \subseteq \Gamma$ і $m \subseteq M$

$$\boldsymbol{g_\gamma}\big(\boldsymbol{\gamma}, \boldsymbol{m}, \boldsymbol{K}(\boldsymbol{m})\big) = \boldsymbol{g}(\boldsymbol{m}), \tag{4.20}$$

тобто $g_\gamma(\gamma, m, u) = g(m)$ при $u = K(m)$.

Умови координованих системи, крім загальних властивостей об'єктів системи S як підмножин нормованих лінійних просторів, базуються на двох передумовах [38]:

- монотонність міжрівневих функцій якості системи

$$\boldsymbol{g(m) \leq g(m')} \text{ при } \boldsymbol{g_i(m) \leq g_i\,(m'), 1 \leq i \leq n, m \subset M, m' \subset M;} \quad (4.21)$$

- існування адитивного уявлення глобальної цільової функції G через задані локальні функції якості G_i, 1≤ i ≤ n, тобто функції

$$\boldsymbol{\widetilde{G}(m,y) = \sum_{i=1}^{n} \sum aiGi\,(mi\,, yi\,, Hi\,(m,y))} \quad (4.22)$$

такої, що глобальні дії, що управляють, оптимальні з точки зору G, одночасно глобально оптимальні по відношенню до $\widetilde{G}$.

Так як функція переваг споживача не має глобального екстремуму, то при орієнтації на індивідуального споживача завдання процесу в загальному випадку доцільно ставити не як завдання оптимізації, а як завдання поліпшення [38]. Тут ефективною є координація шляхом оцінки взаємодій [38], коли підсистеми C_i, $1 \geq i \geq n$, отримують від координатора C_0, крім параметра β, ще й діапазон прогнозних значень зв'язуючих сигналів α, і при вирішенні своїх завдань D(γ) розглядають його як діапазон збурень. Легко помітити, що прогнозування взаємодій можна розглядати як окремий випадок оцінки взаємодій, коли множина обурень є одноелементною.

Таким чином, координацію в процесі з СФ критеріями доцільно будувати на базі оцінки (в окремому випадку - прогнозування) взаємодій. При прогнозуванні взаємодій для кожного $\gamma \subseteq \Gamma$ виділяється множина Uγ, що складається з одного елемента u_γ, тоді завдання і-го локального управляючого елемента є завдання оптимізації на множині M_i с заданим значенням α_γ сполучного входу. Як правило, прогнозування взаємодій використовується паралельно з модифікацією цілей [38]. У цьому випадку значення $\gamma \subseteq \Gamma$ в (4.9) розуміється розширено, а саме:

$$\boldsymbol{\gamma = (\alpha, \beta)}; \quad \Gamma = A \times B, \quad (4.23)$$

де α - прогнозне значення сполучного входу, α ⊆ A, а β - параметр, що конкретизує локальні функції якості $G_{i\gamma} = G_{i\beta}$, β ⊆ B.

Якщо для дворівневої системи задані дві множини A і B такі, що A = K (M), де K описується виразами (4.10), (4.11), а кожне β⊂B визначає локальні цільові функції $g_{i\beta}$, $1 \leq i \leq n$, то для кожного координуючого сигналу γ = (α, β) з A × B і-а локальна задача полягає в мінімізації $g_{i\beta}(mi, \alpha)$ на множині M [38], тобто знаходженні такого значення m_i^{γ}, що

$$\boldsymbol{g_{i\beta}(m_i^{\gamma}) = min_{M_i} g_{i\beta}(m_i, \alpha)}. \quad (4.24)$$

При цьому у фіксованій парі γ = (α, β) з A × B прогноз вважається правильним, якщо

$$\boldsymbol{\alpha = K(m^{\gamma})}, \quad (4.25)$$

де $m^{\gamma} = m_1^{\gamma} \times \ldots \times m_n^{\gamma}$.

Крім того, потрібно виключити ті пари γ = (α, β), для яких прогноз правильний, але управляючий вплив не є глобально оптимальним [39].

ВИСНОВКИ ДО РОЗДІЛУ 4

Однією з найактуальніших проблем масштабної розробки ПЗ є успішне досягнення ефективної координації між командами. Гібридні підходи традиційного та гнучкого розвитку обіцяють поєднувати огляд та передбачуваність довгострокового планування на міжгруповому рівні з гнучкістю та пристосованістю спритного розвитку на командному рівні, однак такі «гібриди» часто виходять з ладу.

Було досліджено як і чому поєднання традиційного планування на рівні між командами та спритного розвитку на рівні команди призводить до неефективної координації у масштабній розробці ПЗ [33]. І виявлено, що основною причиною

неефективної координації між групами є недостатня обізнаність про залежність і те, що сама поінформованість про залежність стримується через неузгодженість планувальних дій на рівні команд та між командами, а саме - уточнення, встановлення пріоритетів, оцінки та розподілу. Знаючи про ці проблеми, великомасштабні гібридні проекти в подібному контексті можуть спробувати краще узгодити свою діяльність з планування на різних рівнях, щоб покращити обізнаність про залежність і, в свою чергу, досягти більш ефективної координації. В даному розділі, на основі всієї проаналізованої інформації, було розроблено адаптовану модель процесу координації гнучкої розробки.

ВИСНОВКИ

Із моменту появи гнучких технологій, які акцентуються на залученні клієнтів, якості продукції, можливості включення мінливих та нових вимог, усвідомленні колективної залученості до створюваного ПП, процес розробки ПЗ зазнав значних та великих змін. У дуже великих, складних та тривалих проектах високу цінність можуть надати гібридні підходи, що передбачають поєднання традиційної планової та гнучкої розробки ПЗ, і дана комбінація видається багатообіцяючою, але проблематичною. За такої моделі процесу успіх програм залежить від здатності проекту керувати цією складністю, а координація в таких умовах є критичною, тому що робота здійснюється одночасно багатьма групами розробки [33].

Дослідження проблеми дало низку розумінь щодо координації між групами в гібридних умовах традиційного планування на високому рівні та гнучкого розвитку безпосередньо в процесах розробки. Було виявлено, що координаційні питання зумовлені в першу чергу тоді, коли між командами не було загального усвідомлення залежностей, і ця відсутність обізнаності про залежність була наслідком невідповідності окремих заходів з планування, які проводились на рівні команд і відповідно на міжкомандному рівні [35]. Також, обізнаність про залежність гальмується розбіжностями у специфікації та розподілі, а неправильна розстановка пріоритетів та оцінка зусиль ще більше посилюють ці проблеми.

Звідси випливає, що створення гібридних проектів можливо, але потребує відповідної управлінської та контрольованої діяльності, а надане розуміння дозволяє глобальним проектам адаптувати кращі практики та технології, що сприяють підвищенню рівня обізнаності про залежність на усіх рівнях.

Дані результати підкреслюють необхідність інституціоналізованих координаційних практик в рамках команд, і така практика повинна передбачати взаємодію установ вищого рівня (такі як ЦК), яка повинна розпочинатися до того, як окремі групи детально планують свої наступні ітерації, вони повинні усунути різницю ефекту зосередженості, деталізації та компетенції, що розділяють учасників

командного та міжгрупового рівнів під час заходів із уточнення, пріоритезації, оцінки та розподілу.

Важливо зазначити, що даною роботою не ставилось за мету узагальнити всі інші параметри гібридних підходів, натомість на меті було створити детальне та глибоке розуміння подій та механізмів у цій справі, щоб запропонувати потужне пояснення за допомогою ретельного збору та аналізу даних.

В даній роботі була також запропонована модель процесу ефективної координації в гібридних умовах (див. рис. 4.4) і важливим є зауваження того, що це не означає, «чим більш узгодженими є планувальні заходи, тим вища обізнаність про залежність та вища ефективність координації». Натомість вона визначає послідовність необхідних умов, які мають відбутися до того, як стане можливою ефективна координація. Таким чином, модель стверджує, що коли планування діяльності не узгоджується, це перешкоджає усвідомленню залежності всіх суб'єктів; а коли немає загальної обізнаності про залежність - координація стає неефективною.

Оскільки сьогодні в гнучких технологіях для великих проектів усі рішення приймаються з використанням практик нарад і спілкування, то для підвищення ефективності процесів прийняття рішень про розробку, забезпечення високого рівня координації та якості ПП необхідно формалізувати ці процеси шляхом розробки математичної моделі (див. рис. 4.5), аби підвищити достовірність, раціональність та оптимальність ухвал, які приймаються при внесені змін в гнучких технологіях. Дана, адаптована для гнучких технологій математична модель, дозволяє побудувати продуктивну організацію координування процесів із критеріями, які слабко формалізуються (що і представляю собою гнучка розробка).

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agile Development at Scale: The Next Frontier. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/Agile_Development_at_Scale (дата звернення 05.10.2020). – Назва з екрана.
2. Key Lessons From Tailoring Agile Methods for Large-Scale Software Development. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/Key_Lessons_Agile_Methods_for_Large-Scale_Software_Development (дата звернення 05.10.2020). – Назва з екрана.
3. Agile manifesto. [Електронний ресурс]. Режим доступу: http://agilemanifesto.org (дата звернення 10.10.2020). – Назва з екрана.
4. Henrik Kniberg. Scrum and XP from the trenches. — C4Media, 2007. — C. 140. — ISBN 978-1-4303-2264-1.
5. Scrum. [Електронний ресурс]. Режим доступу: http://www.mountaingoatsoftware.com/scrum (дата звернення 15.10.2020). – Назва з екрана.
6. Extreme programming. [Електронний ресурс]. Режим доступу: http://www.xprogramming.com/xpmag/whatisxp.htm (дата звернення 17.10.2020). – Назва з екрана.
7. Kanban. [Електронний ресурс]. Режим доступу: https://www.agilealliance.org/kanban (дата звернення 18.10.2020). – Назва з екрана.
8. What is Agile Kanban Methodology? [Електронний ресурс]. Режим доступу: https://www.inflectra.com/methodologies/kanban (дата звернення 26.10.2020). – Назва з екрана.
9. SDLC - Waterfall Model. [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/waterfall_model (дата звернення 28.10.2020). – Назва екрана.

10. Classical Waterfall Model. [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/software-engineering*classical-waterfall-model (дата звернення 29.10.2020). – Назва з екрана.

11. V-Model. [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/sdlc/v_model (дата звернення 30.10.2020). – Назва з екрана.

12. Software Engineering. SDLC V-Model. [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model (дата звернення 30.10.2020). – Назва з екрана.

13. V-Model: An Improvement of Waterfall. [Електронний ресурс]. Режим доступу: https://medium.com/software-engineering-kmitl/v-model (дата звернення 30.10.2020). – Назва з екрана.

14. V-Model in Software Testing. [Електронний ресурс]. Режим доступу: https://www.guru99.com/v-model (дата звернення 30.10.2020). – Назва з екрана

15. Iterative Model: What Is It And When Should You Use It? [Електронний ресурс]. Режим доступу: https://airbrake.io/iterative-model (дата звернення 25.10.2020). – Назва з екрана.

16. SDLC. Iterative Model [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/sdlc/iterative_model (дата звернення 26.10.2020). – Назва з екрана.

17. SDLC. Spiral Model. [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/sdlc/spiral_model (дата звернення 27.10.2020). – Назва з екрана.

18. What Is SDLC Spiral Model? [Електронний ресурс]. Режим доступу: https://www.softwaretestinghelp.com/spiral-model (дата звернення 28.10.2020). – Назва з екрана.

19. Plan-Driven Approaches Are Alive and Kicking in Agile Global Software Development. [Електронний ресурс]. Режим

доступу:https://www.researchgate.net/publication/Plan-Driven_approaches (дата звернення 23.11.2020). – Назва з екрана.

20. Coexisting Plan-driven and Agile Methods: How Tensions Emerge and Are Resolved Completed Research Paper. [Електронний ресурс]. Режим доступу: https://www.researchgate.netPlan-driven_and_Agile_Methods (дата звернення 01.11.2020). – Назва з екрана.

21. Large-Scale Agile Transformation: A Case Study of Transforming Business, Development and Operations. [Електронний ресурс]. Режим доступу: https://link.springer.com/Large-Agile Transformation (дата звернення 23.11.2020). – Назва з екрана.

22. Muhammad, Ali Babar, Alan W. Brown, Ivan Mistrik. Agile software architecture: aligning agile processes and software architectures. - Morgan Kaufmann, 2013. – с. 432. – ISBN 9780124077720.

23. Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/Social_Network_Analysis_Adile (дата звернення 10.11.2020). – Назва з екрана.

24. Застосування архітектурного проектування в гнучких методах розробки програмних продуктів. [Текст] / Харченко О.Г., Боднарчук І.О., Галай І.О., Лісовий В. // Інженерія програмного забезпечення. – 2012. – № 3–4 (11–12). – с. 5–11.

25. Лавріщева К.М. Програмна інженерія. / К.М. Лавріщева –К.: Видавничий дім "Академперіодика", 2008.– 319 с.

26. Харченко О.Г. Проектування архітектури web-застосувань на основі моделі якості. /О.Г. Харченко, І.О. Галай, І.О. Боднарчук, В.В. Яцишин// Інженерія програмного забезпечення. № 4, 2010. – С. 26 – 34.

27. Garlan, David. An Introduction to Software Architecture: Technical Report/ David Garlan, Mary Shaw. – Carnegie Mellon University Pittsburgh, PA, USA, 1994. – 42 p.

28. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб. : Питер, 2010. – 366 с.

29. Руководство Microsoft по проектированию архитектуры приложений (2-е издание) / – Корпорация Майкрософт, 2009. – 528 с.

30. Saaty T. Decision Making with the Analytic Network Process./ Saaty T. Vargas L.// – N.Y.: Springer, 2006. 278 p

31. Harchenko Alexandr, Bodnarchuk Ihor, Halay Iryna. Stability of the Solutions of the Optimization Problem of Software Systems Architecture // Proceeding of VIIth International Scientific and Technical Conference CSIT 2012. pp. 47–48, Lviv, 2012

32. Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. [Електронний ресурс]. Режим доступу:https://www.researchgate.net/publication/Coordinating_Knowledge_Work_in_Multi-Team_Programs (дата звернення 16.11.2020). – Назва з екрана.

33. Studying Onboarding in Distributed Software Teams: A Case Study and Guidelines. [Електронний ресурс]. Режим доступу: https://dl.acm.org/Studying_Onboarding (дата звернення 19.11.2020). – Назва з екрана.

34. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. [Електронний ресурс]. Режим доступу: https://ieeexplore.ieee.org/Coordination Challenges (дата звернення - 03.11.2020). – Назва з екрана.

35. Методы классической и современной теории автоматического управления. Учебник в 3-х т. Т.3: Методы современной теории автоматического управления / Ред. Н.Д. Егупов. М.: Изд-во МГТУ им. Н.Э. Баумана, 2000. 748 с.

36. Ларичев О.И., Мошкович Е.М. Качественные методы принятия решений. Вербальный анализ решений. М.: Наука. Физматлит, 1996. 208 с.

37. Применение теории координации для управления качеством технологических процессов со слабо формализуемыми критериями. [Електронний ресурс]. Режим доступу: https://cyberleninka.ru/article/primenenie-teorii-koordinatsii (дата звернення 18.11.2020). – Назва з екрана.

38. Боулдинг К. Общая теория систем – скелет науки // Исследования по общей теории систем. М.: Прогресс, 1969. С. 106–124.

39. Хакен Г. Синергетика. Иерархия неустойчивостей в самоорганизующихся системах и устройствах. М.: Мир, 1985.

40. Месарович М., Мако Д., Такахара И. Теория иерархических многоуровневых систем. М.: Мир, 1973. 344 с.