МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
"___" _____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Метод забезпечення якості програмних проектів при гібридній технології розробки".

Виконав: студент групи УС-212м Вербовий Роман Андрійович .

Керівник: к.т.н., доцент кафедри КІТ Харченко Олександр Григорович

Нормоконтролер: .Ігор РАЙЧЕВ

Київ – 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Галузь знань, спеціальність: 12 "Інформаційні технології, 122 "Комп'ютерні науки","Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Аліна САВЧЕНКО

" " 2021 р.

ЗАВДАННЯ

на виконання дипломного робота студента

Вербового Романа Андрійовича .

(прізвище, ім'я, по батькові)

1.Тема роботи: "Метод забезпечення якості програмних проектів при гібридній технології розробки" затверджена наказом ректора від «12» жовтня 2021 р. № 2228/ст.

2. Термін виконання проекту: з 12.10.2021р. до 31.12.2021р.

3. Вихідні дані до проекта: рамки оцінки методології "Agile", критерії вибору дослідницького підходу, моделі якості продукуту, продукт та процес архітектурного проектування, гнучкі підходи розробки програмних продуктів.

4. Зміст пояснювальної записки: гнучкі технології проектування програмних продуктів, гібридна технологія розробки програмних продуктів, якість програмних продуктів, використання архітектурного проєктування для підвищення якості програмних продуктів.

5. Перелік обов'язкового графічного матеріалу: «Рисунки гнучкості та зіставлення їх параметрів», «Таблиці аналізу "Scrum", "Lean Development", екстремального програмування », «Моделі теорії адаптивних структур», «Таблиці та рисунки якості систем», «Таблиці прикладів, вимог потреб користувачів», «Рисунки архітектурного інформаційного потоку».

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Визначити та описати основні характеристики гнучких технологій проектування програмних продуктів	12.10.21 –18.10.21	
2	Оцінити рамки методології “Agile”, та проаналізувати та скласти таблиці для таких методологій як: “Scrum”, “Lean Development” та екстримальне програмування (XP)	18.10.21 – 25.10.21	
3	Прояснити та дослідити гібридні методи та технології розробки програмних продуктів	25.10.21 – 31.10.21	
4	Розяснення роботи стандарту ISO 25010, та його область використання, модель якості та його відносинами між моделями	01.11.21 – 15.11.21	
5	Опис архітектрурного проєктування для підвищення якості програмних продуктів	15.11.21 – 23.11.21	
6	Розпис вступу та висновку для кожного з розділів та загального висновку.	23.11.21 – 27.11.21	
7	Створення доповіді та графічного матеріалу до неї	27.11.21 – 08.12.21	
8	Оформлення пояснювальної записки дипломної роботи	08.12.21 – 20.12.21	

Студент-дипломник _____ Роман ВЕРБОВИЙ
(підпис керівника)

Керівник дипломної роботи _____ Олександр ХАРЧЕНКО
(підпис випускниці)

РЕФЕРАТ

Пояснювальна записка до дипломного роботи «Метод забезпечення якості програмних проектів при гібридній технології розробки»: складається із вступу, чотирьох розділів, загальних висновків, списку використаних джерел і містить 94 сторінок тексту, 14 рисунків та 11 таблиці. Список використаних джерел містить 10 найменувань.

Об'єкт розробки – метод забезпечення якості програмних проектів при гібридній технології розробки

Мета роботи – аналіз якості програмних продуктів, модель ISO 25010, використання архітектурного проєкутвання для підвищення якості програмного продукуту.

Об'єкт дослідження – процес забезпечення якості програмних проектів при гібридній технології розробки

Предмет дослідження – Метод забезпечення якості програмних проектів при гібридній технології розробки

Ключові слова – AGILE, SCRUM, ГНУЧКІСТЬ, SCRUM, ISO 25010, GTM, ERP, Kanban, PO, SM, CPO, QA, SRM, SDM .

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Agile – гнучка розробка

Scrum – підхід для гнучкої розробки управління проектами

ASD – техноголія розробки програмних продуктів

ПР – проектні рішення

LD – Lean Development

XP – Extreme Programming

AST – теорія адаптивної структури

Kanban – метод управління розробкою ПЗ

PO – власник продукту

SM – майстер scrum

CPO – головний власник продукту

QA – забезпечення якості

SRM – менеджер якості та обслуговування

SDM – менеджер з надання послуг

GTM – метод обгрунтованої теорії

ERP – система планування ресурсів підприємства

SQUARE – симетричний блочний криптоалгоритм

ПЗ – програмне забезпечення

АЗ – архітектурні знання

ПС – програмна система

ВСТУП

Методи розробки програмного забезпечення Agile розширили концепції забезпечення якості програмного забезпечення, такі як задоволення вимог замовника, перевірки та підтвердження. Гнучкість інноваційно відкриває нові горизонти в галузі забезпечення якості програмного забезпечення. Погляд на гнучкий маніфест виявляє, що гнучка розробка програмного забезпечення — це не лише задоволення потреб клієнтів (тому що навіть методології керовані процесами на це спроможні), але йдеться про відповідність вимогам, що змінюються, аж до рівня розгортання продукту.

Наразі існує достатня кількість літератури, яка прагне описати цей відносно новий набір методологій які змінили спосіб розробки програмного забезпечення. Більшість існуючих робіт від авторів методик і декількох інших практикуючих.

Хоча більшість тих, хто застосовував гнучкі методи у своїх проектах з розробки програмного забезпечення набули переваги які важко ігнорувати у сферах актуальності продукту (результат охоплення нестабільності вимог) та швидкої доставки (результат ітеративного поступового розвитку), деякі з них не приєднуються до цього нового цікавого способу розробки програмного забезпечення через нерозуміння основних концепцій, що лежать в основі гнучких методологій.

Друге занепокоєння було результатом більш ніж трирічних досліджень гнучкої методологічної практики де виявлено, що індивідуальні гнучкі методи, такі як екстремальне програмування, “Scrum” та бережлива розробка тощо, не так відрізняються один від одного. Очевидна відмінність полягає в тому, що люди з різними обчислювальними знаннями які є їх авторами трапляються з різним баченням реального світу. Введена методика оцінювання дозволить виявити схожість по-новому та вирішити проблему сприйняття гнучких методологій. Це також виявляє, що означає якість у гнучкому контексті.

РОЗДІЛ 1
ГНУЧКІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ ПРОГРАМНИХ ПРОДУКТІВ

1.1 .Визначення гнучкої методології (Agile)

Групі методологій розробки програмного забезпечення було дано назву "agile", коли група фахівців з розробки програмного забезпечення зустрілася та створила Agile Alliance (асоціацію практиків з розробки програмного забезпечення, що була створена для формалізації гнучких методологій) у лютому 2001 року поява нової інженерної дисципліни, яка змістила значення процесу розробки програмного забезпечення з механістичного (тобто, керованого процесом і правилами науки) до органічного (тобто, керованого більш м’якими питаннями людей та їх взаємодії). Це передбачає труднощі для інженерних складних програмних систем у робочих середовищах, які є дуже динамічними та непередбачуваними.

Після першої конференції з гнучких методологій у червні 2002 року її учасники узагальнили робоче визначення гнучких методологій як групи процесів розробки програмного забезпечення які є: ітераційними, поступовими, самоорганізуючими та виникаючими. Значення кожного терміну в більш широкому контексті гнучкості показано далі.

1. Ітеративний: слово ітеративне походить від ітерації, яке містить в собі конотації повторення. У випадку гнучких методологій це не просто повторення, а й спроба вирішити програмну проблему шляхом пошуку послідовних наближень до рішення, починаючи з початкового мінімального набору вимог. Це означає, що архітектор або аналітик розробляє повну систему з самомого початку, а потім змінює функціональність кожної підсистеми з кожним новим випуском, оскільки вимоги оновлюються для кожної спроби. Цей підхід на відміну від більш традиційних методів, які намагаються вирішити проблему за один кадр. Ітеративні підходи є більш актуальними для сьогоднішніх проблем розробки програмного забезпечення, які характеризуються високою складністю та швидкою зміною вимогам. З поняттям ітерацій пов'язане поняття поступового розвитку.

2. Поступовий: кожна підсистема розроблена таким чином, що дозволяє збирати більше вимог і використовувати їх для розробки інших підсистем на основі попередніх. Підхід полягає в розподілі зазначеної системи на невеликі підсистеми за функціональністю та додавання нової функціональності з кожним новим випуском. Кожен випуск це повністю перевірена придатна підсистема з обмеженою функціональністю на основі реалізованих специфікацій. По мірі розвитку, корисні функціональні можливості збільшуються до тих пір, поки не буде реалізована повна система.

3. Самоорганізація: цей термін вводить відносно іноземне поняття в управління науковими процесами. Звичайний підхід полягає в тому, щоб організувати команди відповідно до навичок, та відповідних завдань і дати їм можливість розробки звіту для керівництва в ієрархічній структурі. У налаштуваннях гнучкої розробки концепція «самоорганізації» дає команді самостійно організовувати себе для найкращого завершення робочих завдань. Це означає, що реалізація таких питань, як взаємодія в команді, динаміка команди, робочий час, зустрічі про хід роботи, звіти про хід виконання тощо, залишає команді право вибору, як найкраще вони можуть це зробити. Такий підхід є досить ексцентричним щодо того, як навчаються керівники проектів, і він вимагає, щоб керівники проектів разом змінювали парадигму управління. Ця методика вимагає, щоб члени команди поважали один одного і

поводилися професійно, коли мова йде про те, що було вчинено на папері. Іншими словами, керівництво та замовник не повинні отримувати виправдання за невиконання зобов'язань і не повинно бути необгрунтованих запитів про доопрацювання проекту. Роль керівника проекту в такому режимі полягає у сприянні безперебійній роботі команди шляхом зв'язку з вищим керівництвом та усунення перешкод, де це можливо. Таким чином, підхід, що самоорганізовується, передбачає, що між керівництвом проектом та командою з розробки проекту повинна бути чітка комунікативна політика.

4. Виникнення: слово передбачає три речі. По-перше, виходячи з поступового характеру підходу до розвитку, система може виходити з ряду приростів. По-друге, на основі самоорганізуючого характеру у процесі роботи команди формується метод роботи. По-третє, по мірі появи системи і появи методу роботи з'явиться і рамка технологій розвитку. Новий характер гнучких методологій означає, що гнучка розробка програмного забезпечення насправді є досвідом навчання для кожного проекта та залишатиметься досвідом навчання, оскільки кожен проект розглядається по-різному, застосовуючи ітеративні, поступові, самоорганізуючі та новітні методи. На рис. 1 підсумовано теоретичне визначення гнучких методологій.

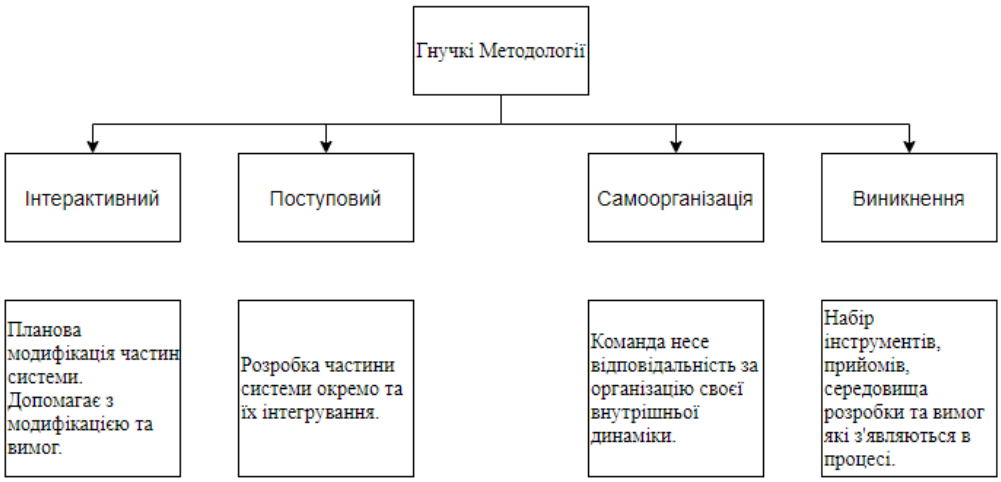

Рис. 1. Визначення гнучкості

Термін "agile" містить у собі конотації гнучкості, спритності, готовності до руху, активності, моторності в русі та регульованості. Кожне з цих слів буде пояснено далі в контексті гнучкості, щоб дати більш точне розуміння видів речей, які робляться в умовах гнучкого розвитку.

- **Гнучкість**: це слово означає, що правила та процеси в умовах гнучкого розвитку можна легко згорнути відповідно до заданих ситуацій, не обов'язково порушуючи їх. Іншими словами, гнучкий спосіб розробки програмного забезпечення дозволяє адаптувати та змінювати.
- **Спритність**: це означає, що при гнучкій розробці програмного забезпечення повинна бути швидка доставка продукту. Зазвичай це робиться шляхом випуску корисних підсистем протягом періоду від одного тижня до чотирьох тижнів. Це дає хороші відшкодування, оскільки замовник почне використовувати систему до її завершення.
- **Готовність до руху**: у гнучкому розвитку загальний намір полягає у зменшенні всіх видів діяльності та матеріалів які можуть або уповільнити швидкість розвитку, або збільшити бюрократизм.
- **Активність**: передбачає фактичне написання коду на відміну від планування, яке інколи займає більшу частину часу на розробку програмного забезпечення.
- **Моторність у русі**: це означає, що в діяльності розробки коду повинно бути достатньо навичок. Навички, про які йдеться, - це розумові навички, які озброюватимуть розробників для програмних завдань та динаміки команди.
- **Регульованість**: це в два рази; по-перше, має бути місце для змін у наборі діяльності та технологій, які є сприятливим до процесу розробки, по-друге, вимоги, код та дизайн/архітектура повинні бути дозволені для зміни на користь замовника.

Гнучкі методології — це легкий, ефективний, з низьким рівнем ризику, гнучкий, передбачуваний, науковий та цікавий спосіб розробки програмного забезпечення. Ці терміни будуть визначені в цьому контексті, щоб дати функціональну перспективу гнучкого розвитку.

• **Легкий:** передбачає мінімізацію всього, що необхідно зробити в процесі розробки (наприклад, документацію, вимоги тощо), щоб збільшити швидкість та ефективність у розробці. Ідея мінімізації документації досі залишається суперечливою, оскільки деякі припускають, що гнучкість не означає жодної документації. Однак такі погляди не є безпідставними, оскільки деякі гнучкі екстремісти висловили конотацію нульової документації, стверджуючи, що код є достатньою документований. По мірі того, як гнучкі методології наближаються до більш високого рівня розвитку документації, що еволюціонують які передбачають надання такої кількості документації, скільки замовник готовий платити за час та гроші.

• **Ефективний:** означає виконати лише таку роботу, яка доставить бажаний продукт з якомога меншими накладними витратами.

• **Низький ризик:** передбачає торгівлю по практичній лінії та залишення невідомого, поки не стане відомо. Насправді всі методології розробки програмного забезпечення розроблені для зменшення ризиків відмови проекту. Часом витрачається багато зусиль на спекулятивне абстрагування проблемного простору в спробі управління ризиком.

• **Передбачуваність:** передбачає, що гнучкі методології базуються на тому, що практикуючі практикують постійно, іншими словами, світ неоднозначності зменшується. Однак це не означає, що планування, дизайн та архітектура програмного забезпечення передбачувані. Це означає, що гнучкість дозволяє розробляти програмне забезпечення найбільш природними способами, яким навчені розробники можуть заздалегідь визначити на основі спеціальних знань.

• **Науковий:** означає, що методологія гнучкої розробки програмного забезпечення базується на обгрунтованих та перевірених наукових принципах. Тим не менш, академічна наука залишається обов'язком продовжувати збирати емпіричні докази про гнучкі процеси, тому що більшість практикуючих, котрі є авторами гнучких методологій, не виникає великого інтересу та часу для проведення такого роду досліджень.

• **Цікавий**: оскільки нарешті розробникам дозволяється робити те, що їм найбільше подобається (тобто витрачати більшу частину часу на написання хорошого коду, який працює). Для розробників гнучкість надає форму свободи бути творчим та інноваційним, не змушуючи клієнта платити за це, замість цього клієнт отримує з цього користь.

Дослідники визначають гнучкий розвиток як протидію 30-річним процесам все більш важких процесів, спрямованим на переоформлення комп'ютерного програмування в інженерію програмного забезпечення, роблячи його таким же керованим і передбачуваним, як і будь-яка інша інженерна дисципліна. З практичної точки зору, гнучкі методології вийшли із загального відкриття серед практиків, що їхня практика повільно відходить від традиційного важкого документообігу та підходів до розробки, орієнтованих на процеси, до більш орієнтованих на людей і менш орієнтованих на документи підходів.

Існує загальне неправильне уявлення про те, що немає планування або мало планування в гнучких процесах. Це пов’язано з тим, що маніфест гнучкого переліку визначає як одне з чотирьох значень перевагу відповіді на зміну після виконання плану. Насправді, планування в гнучких проектах може бути більш точним, ніж у традиційних процесах, воно проводиться суворо для кожного приросту, і з точки зору планування проекту, гнучкі методології забезпечують підхід до зменшення ризику, де найважливішим принципом гнучкого планування є зворотній зв'язок. Інші ж науковці перераховують потенційні ризики як: ризики непорозумінь у функціональних вимогах, ризики глибоко недосконалої архітектури; ризики неприйнятного інтерфейсу користувача; ризики неправильного аналізу та моделей проектування; ризики, коли команда не розуміє обрану технологію тощо.

Зворотній зв'язок отримується шляхом створення робочої версії системи через рівні проміжки часу або з кроком відповідно до попередніх планових зусиль. Окрім подолання найбільш важливих ризиків розвитку програмного забезпечення шляхом поступової розробки, гнучкі методології атакують передумови, що плани, проекти, архітектура та вимоги є передбачуваними і тому можуть бути стабілізовані. "Agile" методології також нападають на передумову повторюваності процесів. Ці два

приміщення є частиною фундаментальних принципів, на яких будуються традиційні методології, і вони також є головними обмеженнями традиційних методологій. Інші ж дослідники розглядають гнучкі методології як виклик для спільноти з розробки програмного забезпечення, яка представляє рух контркультури, який стосується змін з принципово іншої точки зору. Усі гнучкі методології дотримуються чотирьох значень та 12 принципів, як викладено у маніфесті гнучкості.

1.2. Оцінювання якості гнучких процесів

Тож чи можна оцінити забезпечення якості у гнучких процесах?

Це можна зробити за допомогою:

- Надання детальних знань про конкретні питання якості гнучких процесів;
- Визначення інноваційних шляхів підвищення гнучкої якості;
- Визначення специфічних методів якісної якості для конкретних гнучких методологій.

Література показує, що розробка техніки порівняння, метою якої було наведення порівняльного аналізу між якістю моделі розвитку водоспадів та якістю у гнучкій групі методологій отримує такі результати аналізу, що дійсно є гарантія якості у гнучкому розвитку, але це досягається інакше, ніж традиційні процеси, якими він обмежує інструмент це те, що аналіз:

- Виокремлює два основні аспекти управління якістю, а саме забезпечення якості та перевірку та перевірку якості;
- Оглядає інші життєво важливі методи, що застосовуються в гнучких процесах для досягнення більш високого управління якістю;
- Швидке забезпечення якості займає питання якості на крок за межі традиційних підходів до забезпечення якості програмного забезпечення.

Інший виклик методики полягає в тому що, гарантія якості існує у гнучких процесах, але це не дає зрозуміти яким є шлях попереду.

Прихильники, схоже не переживають порівняння між гнучкими та традиційними процесами, оскільки деякі більш ревні " agilist " вважають, що традиційні методи не можуть відповідати гнучким методам у будь-якій ситуації.

Описана оцінка надалі покращується на основі подальшого визначення деяких більш гнучких методів якості, а потім інноваційним шляхом ідентифікує гнучкі практичні процеси, що відповідають кожній методиці. Внесок цієї оцінки є подальшим виявленням можливих практик, які можна вдосконалити для покращення високої якості досягнень якими користуються гнучкі процеси.

Параметри, які визначають якість програмного забезпечення з виду високого рівня, можуть бути досить абстрактними. Однак запропонована методика підбирає кожен з параметрів і визначає відповідні гнучкі методи, що реалізують параметр так чи інакше. Можливі вдосконалення існуючої практики були запропоновані шляхом аналізу способів роботи гнучких практиків. Важливе значення для цього виду аналізу має огляд деяких інтуїтивних практик, які розробники зазвичай застосовують, які можуть бути не задокументовані.

Справа в тому, що розробники розповідають свої історії успіху на різних професійних форумах, а деякі підказки з таких роздумів були використані в цій техніці, не дотримуючись жодної формальної методології збору даних. Дослідники вважають, що збір неофіційних необроблених даних врівноважує факти, особливо у випадках, коли розробники говорять про свою практику. Після того, як дані будуть зібрані формально, з'являється чимало забобонів і упереджень, і для збалансування фактів знадобиться застосувати інші методи дослідження. У (табл. 1.1 та 1.2) узагальнено підхід до оцінювання.

Таблиця 1.1

Зіставлення параметрів якості програмного забезпечення для гнучких методів

Параметри якості програмного забезпечення	**Гнучкі методи**	**Можливі вдосконалення**
Правильність	Напишіть код з мінімальних вимог. Специфікація отримується шляхом прямого спілкування із замовником. Клієнт може змінювати технічні характеристики. Тест-орієнтований розвиток.	Розглянемо можливість використання формальних специфікацій у гнучкій розробці, Можливе використання загальних сценаріїв для визначення вимог (зауважте, що деякі команди розробників вже використовують це).
Міцність	Не сприймається безпосередньо в умовах гнучкого розвитку.	Включайте можливі екстремальні умови в вимоги.
Розширюваність	Загальна особливість усіх розроблених ОО додатків. Акцент робиться на технічну досконалість та гарний дизайн. Акцент також на досягненні найкращої архітектури.	Використання методів моделювання архітектури програмного забезпечення.
Багаторазовість	Загальна особливість усіх розроблених ОО додатків. Існують деякі аргументи проти повторного використання гнучких продуктів.	Розробка моделей для гнучких додатків.
Сумісність	Загальна особливість усіх розроблених ОО додатків.	Чи можна додавати додаткові функції заради сумісності, навіть якщо вони можуть не знадобитися? Це може суперечити принципу простоти.
Ефективність	Застосовуйте хороші стандарти кодування.	Заохочуйте конструкції на основі найбільш ефективних алгоритмів
Переносимість	Практика постійної інтеграції в екстремальне програмування.	Деякі гнучкі методи безпосередньо не вирішують проблеми розгортання продукту. Вирішити це могло б на користь гнучкості.
Своєчасність	Найсильніша точка гнучкості, Короткі цикли, швидка доставка тощо.	---
Цілісність	Не сприймається безпосередньо в умовах гнучкого розвитку.	---
Перевіреність	Тестова розробка - ще одна сила гнучкості.	---
Простота використання	Оскільки замовник є частиною команди, а клієнти часто дають відгуки, вони, швидше за все, порекомендують просту у використанні систему.	Дизайн для найменш кваліфікованого користувача в організації.

У формальному управлінні якістю програмного забезпечення діяльність із забезпечення якості виконується шляхом забезпечення того, щоб кожен із параметрів, перелічених у (табл. 1.1), певною мірою був дотриманий у життєвому циклі розробки програмного забезпечення відповідного процесу. Коротке визначення кожного з цих параметрів:

- **Правильність**: здатність системи працювати відповідно до визначених специфікацій;
- **Надійність**: відповідна продуктивність системи в екстремальних умовах;
- **Розширюваність**: система, яка легко адаптується до нових специфікацій;
- **Багаторазовість**: програмне забезпечення, що складається з елементів, які можна використовувати для побудови різних додатків;
- **Сумісність**: програмне забезпечення, яке складається з елементів, які легко поєднуються з іншими елементами;
- **Ефективність**: здатність системи ставити якомога менше вимог до апаратних ресурсів, таких як пам'ять, пропускна здатність, що використовуються для зв'язку та час процесора;
- **Переносимість**: простота установки програмного продукту на різних апаратних та програмних платформах;
- **Своєчасність**: звільнення програмного забезпечення перед або саме тоді, коли воно потребує користувачів;
- **Цілісність**: наскільки добре програмне забезпечення захищає свої програми та дані від несанкціонованого доступу;
- **Перевіреність**: як легко протестувати систему;
- **Простота використання**: простота, з якою люди різних класів можуть вивчати та користуватися програмним забезпеченням.

1.3. Порівняльне оцінювання гнучких технологій

Більш масштабний вигляд гнучких процесів призводить до уявлення про те, що гнучкі методи — це група процесів, що скоротили часові рамки розробки програмних

систем та запровадили інноваційні методи для прийняття швидких мінливих бізнес-вимог. З часом ці відносно нові методи повинні перерости у зрілі стандарти інженерних програм.

Гнучка перспектива моделювання програмних процесів полягає в тому, що незалежно від того, чи використовуються підходи до моделювання, формальні чи неофіційні, ідея полягає в застосуванні методів моделювання таким чином, щоб документація була мінімізована і простота потрібної системи ― доброчесність. Моделювання гнучкого способу призвело до проривів у застосуванні гнучких методів до розвитку великих систем.

Гнучкий підхід до управління програмними проектами заснований на наданні більшої цінності розробникам, ніж процесу. Це означає, що керівництво повинно прагнути зробити сприятливе середовище розвитку. Замість того, щоб турбуватися про обчислення критичного шляху та графіки керівник проекту повинен полегшити спілкування віч-на-віч і простіші способи отримання зворотного зв'язку про хід проекту. У гнучкому розвитку потрібно бути оптимістичним щодо людей і вважати, що вони означають добро отже, дати їм простір для розробки найкращого способу виконання своїх завдань. Це також гнучка стратегія ― довіряти, що люди приймуть правильні професійні рішення щодо своєї роботи та забезпечать представництво замовника в колективі протягом усього проекту.

1.3.1. Рамки оцінки методології "Agile"

Усі гнучкі методології мають надзвичайну схожість між своїми процесами, оскільки вони засновані на чотирьох гнучких значеннях та 12 принципах. Цікаво зауважити, що навіть автори гнучких методологій більше не наголошують на своїх методологічних межах і застосовуватимуть практики інших гнучких методологій, якщо вони відповідають конкретній ситуації. Детальний огляд гнучких методологій показує, що гнучкі процеси вирішують ті самі проблеми, використовуючи різні моделі реального життя.

Методика оцінки, представлена в цій главі, показує, наприклад, що “Lean Development” (LD) розглядає розробку програмного забезпечення з використанням метафори виготовлення та виробу. “Scrum” розглядає процеси розробки програмного забезпечення за допомогою метафори керуючої інженерії. Екстремальне програмування (XP) розглядає діяльність з розробки програмного забезпечення як соціальну діяльність, де розробники сидять разом. Розробка адаптивних систем (ASD) розглядає проекти програмного забезпечення з погляду теорії складних самоадаптаційних систем.

У (табл. з 1.2 до 1.5) узагальнено аналіз гнучких методологій. Для ілюстрації методики оцінки було обрано лише декілька існуючих гнучких методологій. У першому стовпці зліва на (табл. 1.2, 1.3 та 1.4) перераховані деякі елементи методології, які були обрані для представлення деталей методології. Існує багато суб'єктивності навколо вибору елементів методології. Не в межах цієї глави представлена повна таксономія методологій. Тому елементи, використані тут, були обрані для виявлення подібності між різними гнучкими методологіями. Важливість виявлення цих подібностей полягає в тому, щоб озброїти розробників, які потрапили в гнучкі методичні джунглі, задаючись питанням, яку методологію обрати.

Хоча методологія, що використовується у проекті з розробки програмного забезпечення, не може безпосередньо призвести до успіху проекту і, можливо, не призведе до виробництва високоякісного продукту, використання неправильної методології призведе до збою проекту. Отже, є розумність вибору правильного і відповідного процесу. Більшість організацій може не дозволити собі розкіш використання різних методологій для кожного проекту, хоча це було б ідеально для більших досягнень. Це також звучить непрактично, щоб мати робочу силу, яка володіє багатьма методологіями.

Дотримуватися однієї методології та очікувати, що її буде достатньо для всіх проектів, також було б наївно. Таким чином, ця методика оцінювання дає організаціям з розробки програмного забезпечення інноваційну кмітливість для адаптації процесу їх розробки відповідно до загальних практик різних гнучких

методологій. Перевага полягає у використанні багатьох методологій без пов'язаних з цим витрат на їх придбання.

Необхідно детально розібрати кожну гнучку методологію, яка буде проаналізована, щоб виявити основні принципи методології. Ця методика дає основні деталі щодо того, чому методологія була розроблена в першу чергу. Відповідь на це запитання дозволить виявити основні проблеми, які викликають занепокоєння методологією. Тоді потенційний користувач методології вирішить, чи відповідає така сфера їхнього проекту. Виявлення проблем, які методологія має намір вирішити, є ще одним питанням цієї оцінки. Деякі методології мають загальний ухил до вирішення технічних проблем у процесі розробки (тобто екстремальне програмування стосується таких питань як і коли перевірити код). Існують і інші гнучкі методології, які вирішують проблеми управління проектами (тобто Scrum займається такими питаннями, як ефективне спілкування в рамках проекту). Можливо, існують інші гнучкі методології, які вирішують цілий ряд проблем.

Оцінка кожної методології також повинна виявити, які види діяльності та практики є переважаючими в методології. Це повинно допомогти потенційним користувачам методології визначити практики, які можуть бути відповідні їх ситуації. Ця методика оцінки показує, що деякі практики з різних методологій насправді відповідають однаковим гнучким принципам, і розробники вирішуватимуть, які практики є можливими в їхній ситуації.

Отже, наслідком є те, що на рівні реалізації стає неважливим, яка гнучка методологія використовується. Інший аспект гнучких методологій, розкритих за допомогою цієї методики оцінювання, - це те, що методологія надає в кінці проекту. Коли розробник шукає методологію вони, як правило, мають певні очікування щодо того, що вони хочуть, як вихід з методології.

Отже, якщо результат роботи методології не є чітко зрозумілим, це може спричинити проблеми. Наприклад, якщо розробник розраховує, що використання методології призведе до доставки коду, і все ж мета методології насправді полягає у створенні набору артефактів дизайну, таких як ті, що доставляються за допомогою гнучкого моделювання, це може призвести до деяких проблем. Також, ця методика

оцінювання розкриває доменні знання автора методики. На цьому етапі аналізу не потрібно згадувати імен авторів, а просто констатувати їх доменну експертизу. У таблицях з 2 до 4 наведено резюме аналізу конкретних гнучких методологій, щоб проілюструвати, як ця методика аналізу може бути використана для будь-яких гнучких методологій.

1.3.2. Аналіз “Scrum”

Scrum використовувався порівняно довший період, ніж інші гнучкі методики. Scrum, поряд з XP є однією з найбільш широко використовуваних гнучких методологій. Основна увага Scrum робиться 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", що очевидно, неможливо. Щоб вирішити проблему визначених і повторюваних процесів, Scrum розділяє проект на ітерації (які називаються спринтами). Перш ніж розпочати спринт, для цього спринту визначається необхідна функціональність, і команді залишається його доставити. Справа в стабілізації вимог під час спринту. Scrum наголошує на концепціях управління проектами, хоча деякі можуть стверджувати, що Scrum настільки ж технічний, як і XP. Термін Scrum запозичений у Регбі: "Scrum виникає, коли гравці кожної команди тісно збиваються ... у спробі просунутись в ігрових умовах". У (табл. 1.2) показано застосування методики аналізу до Scrum.

Таблиця 1.2

Аналіз методології Scrum

Елементи	Опис
Метафора реального життя	Техніка управління.
Фокус	Управління процесом розвитку.
Область застосування	Команд менше 10, але масштабується для більших команд.
Процес	Фаза 1: планування, відставання продукту та дизайн. Фаза 2: відставання спринту, спринт. Фаза 3: тестування системи, інтеграція, документація та звільнення.
Результати	Робоча система.
Прийоми та інструменти	Спринт, відставання в scrum (випадки використання письма).
Автор методики (два)	1. Розробник програмного забезпечення, менеджер продуктів та галузевий консультант. 2. Розроблені мобільні додатки на відкритій платформі технологій. Розробник компонентних технологій. Архітектор сучасних систем робочого процесу в Інтернеті.

1.3.3. Аналіз методології “Lean Development”

Розробка програмного забезпечення, метод розробки динамічних систем та Scrum - це скоріше набір практик управління проектами, ніж певний процес. Він був розроблений Бобом Шареттом, і він спирається на успіх, яке (LD) здобуло в автомобільній промисловості в 1980-х. Хоча інші гнучкі методології прагнуть змінити процес розвитку, Шарет вважає, що щоб бути по-справжньому гнучким, необхідно замірити, як в компанії працюють згори вниз. Перша розробка орієнтована на те, щоб змінити спосіб генеральних директорів розглядати зміни щодо управління проектами. LD базується на бережливому мисленні, походження якого виявляється у

бурежливому виробництві, розпочатому виробничою компанією Toyota Automotive. У таблиці 3 показано застосування методики аналізу на LD.

Таблиця 1.3

Аналіз методології “Lean Development”

Елементи	Опис
Метафора реального життя	Виробництво та розробка продукції.
Фокус	Управління змінами. Управління проектами.
Область застосування	Немає конкретного розміру команди.
Процес	Не має процесу.
Результати	Надає знання для управління проектами.
Прийоми та інструменти	Тонкі технології виготовлення.
Автор методики (два)	Науковий співробітник Центру морських підводних систем США, автор книг та робіт з інженерної програми, дорадча рада з управління проектами.

1.3.4. Аналіз екстремального програмування (XP)

Екстремальне програмування (XP) ― це легка методологія для невеликих та середніх команд, що розробляють програмне забезпечення на основі розпливчастих або швидко мінливих вимог. У другій версії XP Бек розширив визначення XP, щоб включити розмір команди та обмеження програмного забезпечення наступним чином:

- **XP легкий**: робити лише те, що потрібно зробити, щоб створити цінність для клієнта;

- **XP адаптується до розпливчастих і швидко мінливих вимог**: досвід показує, що XP можна успішно використовувати навіть для проектів зі стабільними вимогами;
- **XP вирішує обмеження в розробці програмного забезпечення**: він не стосується безпосередньо управління портфелем проектів, фінансовими проблемами проекту, операціями, маркетингом або продажами.

Таблиця 1.4

Аналіз екстремального програмування

Елементи	Опис
Метафора реального життя	Соціальна діяльність, де розробники сидять разом.
Фокус	Технічні аспекти розробки програмного забезпечення.
Область застосування	Менше десяти розробників в кімнаті. Масштабується до більших команд.
Процес	Фаза 1: Написання розповідей користувачів. Фаза 2: Оцінка зусиль, визначення пріоритетності історії. Фаза 3: Кодування, тестування, тестування інтеграції. Фаза 4: Малий випуск. Фаза 5: оновлений випуск. Фаза 6: Остаточний реліз.
Результати	Робоча система.
Прийоми та інструменти	Парне програмування, рефакторинг, тестова розробка, безперервна інтеграція, метафора системи.
Автор методики (два)	1. Розробник програмного забезпечення . Сильно віруюча у спілкуванні, роздумах та новаторстві. Шаблон для програмного забезпечення. Тест-перша розробка. 2. Розробник програмного забезпечення. Директор з досліджень та розробок. Розробив Wiki. Розроблена основа для інтегрованого тестування.

Розробка програмного забезпечення за допомогою XP починається зі створення історій замовником для опису функціональності програмного забезпечення. Ці історії це невеликі одиниці функціональності, які потребують близько тижня-двох для кодування та тестування. Програмісти надають оцінку для оповідань, замовник вирішує, виходячи із вартості та вартості, які історії першими робити. Розробка робиться ітераційно та поступово. Кожні два тижні команда програмування доставляє робочі історії замовнику. Тоді замовник вибирає ще два тижні роботи. Система зростає у функціональності, по частинах, та керується замовником. У таблиці 4 показано застосування методики аналізу до XP.Кожен з методологічних елементів, представлених у (табл. 1.2 до 1.4), буде визначений у контексті цього підходу до аналізу.

Метафора реальної життєдіяльності: цей елемент посилається на фундаментальну модель або метафору та обставини, що викликали початкове уявлення про методологію. Наприклад, спостереження за процесом, за яким мурахи збирають мурашник, може викликати ідею застосувати той самий процес до розробки програмного забезпечення.

Методологічний фокус: у центрі уваги методології є конкретні особливості процесу розробки програмного забезпечення, на який спрямована методика. Наприклад, гнучке моделювання орієнтоване на аспекти дизайну процесу розробки програмного забезпечення, а також розглядає питання, як моделювати великі та складні проекти гнучким способом.

Методика області застосування: цей елемент окреслює деталі, до яких прописано рамки розробки методології. Саме тут методологія визначає, що вона охоплює в рамках проекту. Важливість цього параметра полягає в тому, щоб допомогти користувачеві визначити перелік завдань, якими методологія допоможе керувати. Пам'ятайте, що методологія не робить все, а просто дає рекомендації, які допомагають в управлінні проектом. Обсяг проекту розробки програмного забезпечення є актуальним при визначенні чисельності команди.

Методичний процес: цей параметр описує, як методологія моделює реальність. Модель може бути відображена в життєвому циклі чи процесі розробки

методології. Модель забезпечує засіб комунікації, фіксує суть проблеми чи конструкції та дає розуміння проблемної області. Важливість цього параметра полягає в тому, що він дає користувачеві реальне світосприйняття низки заходів, які проводяться в процесі розробки.

Результати методології: цей параметр визначає форму результатів, яку слід очікувати від методології. Наприклад, якщо сьогодні організація придбала методологію розвитку, вона отримає код від застосування методології, або отримає деякі документи тощо. Кожна гнучка методологія дасть різні результати, тому користувач може обрати методику, яка дає їм необхідний результат.

Прийоми та інструменти методології: цей параметр допомагає користувачеві визначити методи та інструменти, застосовні до методології. Інструменти можуть бути програмними, які можна використовувати для автоматизації деяких завдань у процесі розробки, або вони можуть бути такими ж простими, як дошки та фліп-схеми. Насправді саме використання інструментів робить реалізацію методології приємною. Тому організації, як правило, витрачають багато грошей на придбання інструментів та навчання персоналу інструментам. З розвитком технології та появою нових інструментів зазвичай проводиться більше придбань та навчання. Однак більшість гнучких методологій не визначають інструменти, а більшість гнучких практиків використовують інструменти з відкритим кодом, що зменшує потенційні витрати на програмні засоби. Кожна методологія має свої прийоми, які можуть бути актуальними або не стосуватися існуючої проблеми. Прикладами прийомів екстремального програмування можуть бути парне програмуванн. Потім користувач аналізує ці методи стосовно даного проекту, щоб визначити необхідність цих методів та включити варіанти, які будуть частиною адаптації методології.

Автор методики: цей параметр визначає доменні знання автора методології. Вигода від цього полягає в з'ясуванні передумови, з якої була задумана методологія. Не потрібно згадувати прізвище автора чи детальну біографію автора методики. Таблиця 5 підсумовує етап аналізу, коли всі практики об'єднуються та подібні практики виявляються в різних методологіях. Таблиця 5 класифікує практики,

використовуючи надписи 1, 2, 3, 4 і 5. Практики з тим самим суперскриптом реалізують той самий гнучкий принцип:

• "1" - це практика, яка займається питаннями планування, такими як збір вимог;

• "2" - це практика, яка стосується покращення якості з точки зору задоволення нестабільних вимог;

• "3" представляє практику, яка полегшує вільну співпрацю розробників, ефективне спілкування, розширення питань щодо прийняття рішень та динаміки команди;

• "4" - це практика, яка стосується швидкої доставки товару;

• "5" представляє практику, яка стосується гнучкої властивості забезпечення якості забезпечення постійного вдосконалення продукту до розгортання. Після виявлення подібних практик розробники можуть вирішити вибрати та адаптувати деякі практики до свого середовища відповідно до релевантності та пріоритетів проекту та замовника.

Таблиця 1.5

Визначення подібності між методологіями

	Практики
XP	Процес планування(1), невеликі випуски(2), метафора, тестова розробка(2), пріоритетність сюжетів(3), колективна власність(3), парне програмування(3), сорокагодинний робочий тиждень(3), замовлення на місці(4), рефакторинг(5), простий дизайн(5) та безперервна інтеграція(5).
LD	Усуньте відходи(1), мінімізуйте товарні запаси(1), максимізуйте потік(2), витягніть з попиту(2), відповідайте вимогам замовника(2), забороніть місцеву оптимізацію(2), надайте можливості працівникам(3), зробіть це правильно вперше(4), співпрацюйте з постачальниками(4) та створіть культуру постійного вдосконалення(5).
Scrum	Вимоги до захоплення як відставання продукту (1), тридцятиденний спринт без змін під час спринту(2), зустрічі Scrum (3), самоорганізуючих команд (3) та зустрічі планування спринту(4).

1.4. Огляд проблем і протиріч “Agile”

1.4.1. Спільна розробка програмного забезпечення

Проблеми, які виконуються аналогічно серед різних методологій розробки програмного забезпечення і ці більшість процесів розробки програмного забезпечення, що застосовуються сьогодні, включають деякі з наступних заходів: планування, оцінка та планування завдань, проектування, кодування та тестування, розгортання та обслуговування. Залежно від різних процесів залежить послідовність, яка дотримується при здійсненні кожної з фаз, і рівень деталізації, на якому здійснюється кожна фаза.

Деякі методології можуть реалізовувати всі дії, а деякі часткові методології можуть спеціалізуватися лише на кількох. Інша відмінність полягає у тому, як процес оцінює людей, які беруть участь у розробці, і яку цінність надає замовнику стосовно того, що потрібно зробити. Ці відмінності відзначають основні межі методологій розробки програмного забезпечення.

1.4.2. Гнучкий розвиток вищого рівня

Питання, які своєрідно виконуються за допомогою гнучких методологій ти експерти з розробки програмного забезпечення, що мають практичний досвід у цій галузі, мають багато знань, які можна кваліфікувати як мовчазні знання через те, що вони здобуваються через практику і не записуються ні в якій формі. Мовчазні знання важко піддаються кількісній оцінці, тому ця концепція залишається досить суб’єктивною у застосуванні гнучких методологій.

Однак сила використання мовчазних знань полягає в командному дусі, який довіряє експертам робити те, що вони найкраще знають, у межах своєї професійної етики. Це насправді і відрізняє «рухливий рух» від інших процесів розвитку. Ще одне гаряче питання про гнучкий розвиток ─ це концепція самоорганізаційних команд. Ця

концепція означає, що для гнучких команд з розвитку дозволено організовувати себе найкращим чином для досягнення поставлених цілей.

В результаті застосування цієї концепції управління гнучких проектів стає відмінним від традиційних підходів до управління проектами. Роль керівника проекту стає скоріше фасилітатором, ніж контролером. Ця концепція була досить суперечливою з тих пір, як почалися гнучкі методики. Основна причина суперечки полягає в тому, що традиційні методології завжди пов'язують документацію з належним плануванням, забезпеченням якості програмного забезпечення, розгортанням, навчанням користувачів, технічним обслуговуванням тощо.

Однак, методологи Agile вважають, що документація повинна бути мінімальною через пов'язані з цим витрати. У гнучких методологіях загальна думка полягає в тому, що правильно записаний код достатній для обслуговування. Спочатку тестова методика, яка спочатку була практикою XP, а зараз широко застосовується в інших гнучких методологіях, є ще однією своєрідною гнучкою практикою (хоча її походження може бути від попередніх процесів). Перша методика тестування - це підхід до розробки програмного забезпечення, який реалізує розробку програмного забезпечення за допомогою написання тестів для кожної історії до написання коду. Потім тестовий код складається з конструктивних артефактів і замінює потребу в конструкторських діаграмах тощо.

1.5. Напрями розвитку “Agile”

Швидка методологія, безумовно, рухається до більш високого рівня розвитку завдяки ряду речей. Перший внесок у гнучкий розвиток — це наявність вичерпних джерел простої описової та аналітичної інформації про гнучкі методології. Другий внесок у гнучкий розвиток — зростання інтересу до академічних досліджень до гнучкості, в результаті якого було зібрано та науково проаналізовано багато емпіричних даних для підтвердження та спростування анекдотичних даних про рухливі процеси. Третій внесок у гнучкий розвиток - це масовий обмін практичним

досвідом серед різних практиків, які беруть участь у розробці гнучкого програмного забезпечення.

Загальний напрямок рухливого руху, схоже, спрямований на все більший попит на впровадження гнучких практик з боку великих організацій, які традиційно асоціюються з традиційними процесами. Повідомлялося про більш високі вимоги до гнучких консультацій та навчання, оскільки все більше організацій застосовують гнучкі практики розвитку. Це, безумовно, пройде довгий шлях у підвищенні розвитку технологічних технологій.

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі було представлено огляд гнучких методологій. Основна увага полягала в тому, щоб забезпечити усвідомлений огляд гнучких методологій, які включали всебічне визначення того, що таке гнучкість та гнучка якість Підхід до визначення повинен був дати теоретичне визначення, яке є точкою зору тих, хто по-філософськи вивчає гнучкі методології та практичне визначення, яке є перспективою тих, хто знаходиться на робочих поверхах розвитку, і контекстуальне визначення, яке є перспективою, що базується на різних контекстах діяльності процесу розробки програмного забезпечення. Для підвищення розуміння гнучких процесів була представлена модель аналізу.

Філософія цієї методики полягає в тому, щоб проникнути глибоко в кожну задану методологію та розкрити основні цінності, принципи та практики методології, щоб порівняти спільну діяльність серед різних гнучких процесів. Метою проведення такого аналізу є створення методики досягти рівноваги між цими двома крайнощами: «загубитися у гнучких джунглях методології та дотримуватися однієї методології». Перевага використання цього методу аналізу полягає в досягненні більш глибокого розуміння всіх проворних методологій, що аналізуються. Це повинно закласти грунт для навчання та прийняття гнучких методологій з загальної точки зору, а не турбуватися про окремі гнучкі методології.

РОЗДІЛ 2
ГІБРИДНА ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

2.1. Гібридні методи проти співіснуючих методів

Гібридні підходи до розробки програмного забезпечення охоплюють елементи методів, керованих планом (наприклад, водоспад або V-модель) і гнучких методів. Гібридний підхід може поєднувати елементи двох або більше різних методів SD, наприклад різні

- **характеристики** (наприклад, бюрократичні процеси з високою формалізацією методів, керованих планом, у поєднанні з кооперативними соціальними діями між людьми, які використовують гнучкі методи),
- **практики** (наприклад, обширна фаза проектування методів, керованих планом, у поєднанні з короткими кроками гнучких методів)
- **ролі** (наприклад, керовані планом менеджери проектів у поєднанні з agile scrum майстри та власники продуктів). Навмисно вибираючи елементи як гнучкі, так і керовані планом для поєднання корисних практик методів, організації формують специфічний гібридний підхід уздовж континууму між обома методами, як показано на (рис. 2.1).

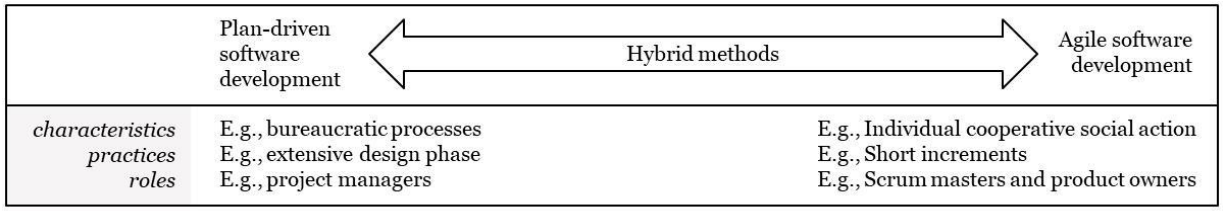

Рис. 2.1 Континуум планових і гнучких методів розробки програмного забезпечення

Комбінація елементів швидкості та плану призводить до співіснування обох методів, які можуть доповнювати один одного. Однак співіснування може також викликати напруженість між різними або навіть протилежними елементами планових і гнучких методів. Напруга визначається як «елементи, які здаються логічними окремо, але непослідовними, навіть абсурдними, коли їх поєднують». Загалом, напруженість виникає в результаті двох контрастних, але суперечливих стилів управління. Особливо на рівні команди, коли команди співпрацюють одна з одною, протилежні переконання щодо гнучких і керованих планом методів стикаються. Наприклад, гнучкі методи заохочують команди до самоорганізації своєї роботи, тоді як команди, які використовують підходи, орієнтовані на план, звикли виконувати накази. Вивчаючи напруженість між протилежними елементами методів, керованих планом, і методів, що керуються гнучкістю, ми зосереджуємо наше розгляд на співіснуванні методів, а не на вигідних комбінаціях (тобто гібридних підходах).

2.2. Теорія адаптивної структури

Теорія адаптивної структури (AST) пропонує підхід до дослідження соціальних аспектів групових процесів. Соціальні аспекти визначають технологію, яку використовують групи та соціальні структури, на які впливає груповий вибір. AST досліджує взаємний вплив технологій і соціальних процесів, щоб проаналізувати, як групи пристосовуються до структур і, у свою чергу, як технології впливають на групи. Походячи з теорії структури, AST досліджує взаємодію структурованість, «процес впровадження правил, ресурсів та інших структур у дію» та привласнення, що визначається як застосування структур у окремому контексті. Попередні дослідження гнучких методів використовували AST різними способами для розуміння ролі

соціальних структур у SD. вивчали адаптацію методів agile на проектному та організаційному рівні. Розширюючи свої дослідження визначили традиційний процес фінансування IT як складність на рівні проекту при прийнятті гнучких методів. досліджував адаптацію agile методів за організаційними особливостями, викликаними їх взаємодією, і навпаки. Насамперед, AST використовувався для досліджень на рівні команди, наприклад, для аналізу віртуальних команд. У дослідженні ми знову зосереджуємось на прийнятті гнучких методів на рівні команди. таким чином,

Використовуємо AST для вивчення структури команд і ролей команд, використовуючи сумісні методи. Співіснування керованих планом і гнучких методів виникає внаслідок прийняття організацією гнучких методів. Спираючись на AST, ми прагнемо зрозуміти, як групи самоорганізуються і як заохочуються різні форми соціальної взаємодії. Ми прагнемо отримати нове уявлення про співпрацю в середовищі співіснуючих планових і гнучких методів. На (рис. 2.2) зображена структура.

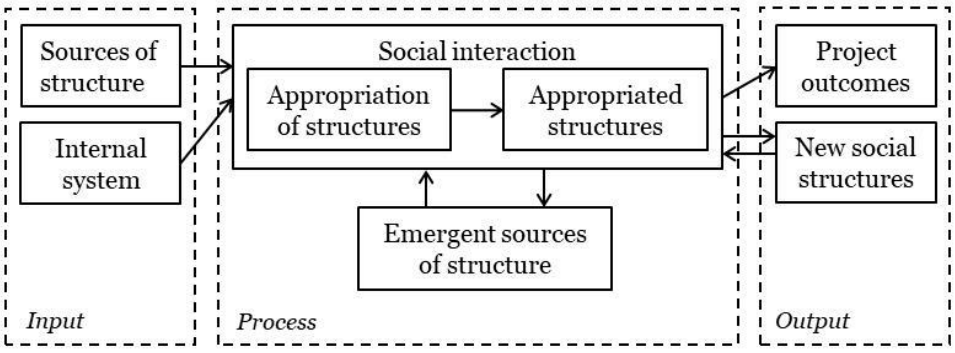

Рис. 2.2 Модель теорії адаптивної структури

Модель AST складається з трьох елементів: входу, процесу та виходу. Вхідна фаза включає джерела структури та внутрішню систему команди. Джерелами структури нашої моделі є керовані планом і гнучкі методи.

Структура методів, керованих планом характеризується негнучким послідовним процесом, який «виробляє рішення проблем, які вже змінилися».

Проекти, керовані планом, вимагають формальної ієрархії, чітко документованих процесів і підтримувати дисципліну планування, щоб уникнути ризиків.

Структура гнучких методів містить цінності та притаманні правила та можливості. На відміну від методів, керованих планом, вони підкреслюють гнучкість та автономність. Щоб заохочувати автономію, повноваження делегуються команді шляхом сприяння самоорганізації. Гнучкі методи характеризуються підвищеною чутливістю до мінливих вимог, більш тісною взаємодією з клієнтами та ітеративним підходом до розробки.

Внутрішня система команди описує моделі взаємодії на рівні команди. Наприклад, досвід членів команди спільної роботи або з використанням agile методів або їх індивідуальний стиль керівництва.

Фаза процесу описує соціальну взаємодію між командами з різними джерелами структури (наприклад, керованими планом або гнучкими методами), які створюють виникаючі джерела структури і впливають на соціальну взаємодію в свою чергу.

Соціальна взаємодія описує привласнення структур що призводять до привласнені структури. Привласнення відноситься до реалізації прикладних соціальних структур у конкретному контексті. AST пропонує чотири характеристики привласнення: ходи присвоєння, вірність присвоєння, інструментальне використання, і ставлення до привласнення.

Ходи присвоєння описують, як команди засвоюють структури (наприклад, як метод agile використовується в команді). Вірність описує узгодженість структури з її духом (наприклад, ступінь, до якої метод agile узгоджується з загальними рекомендаціями). Інструментальне використання визначає мету привласнення. Нарешті, ставлення до привласнення відноситься до «сприйняття та почуттів користувачів щодо структури». Адаптація призводить до привласнені структури яким сприяє соціальна взаємодія (наприклад, рішення адаптувати методи до потреб гнучких команд). Виникаючі джерела будови виникають внаслідок застосування та перебудови конструкцій. Наприклад, схожість із методами agile перешкоджає процесам соціальної взаємодії (наприклад, команди відмовляються адаптуватися до agile методів).

Третій аспект моделі описує результат процесу соціальної взаємодії. Результатом соціальної взаємодії єрезультати проекту, пов’язаних з графіком, бюджетом або обсягом. Відбудовуючи існуючі споруди, нові соціальні структури з'являються такі, що можуть включати джерела структури або внутрішню систему (наприклад, вводити нові стилі керівництва) і впливати на соціальну взаємодію.

2.3. Критерії вибору дослідницького підходу

Застосуємо дослідницький підхід до вивчення кількох випадків, щоб проаналізувати співіснування керованих планом і гнучких методів, які з’являються у співпраці між гнучкими та неспритними командами. Конструкція з кількома випадками дозволяє нам проводити перехресний пошук шаблонів і підходить для відповіді на питання «як» і «чому» шляхом дослідження явища в контексті його реального життя. На основі Паттона ми цілеспрямовано та інформативно відбирали наші кейси, що означало, що кейси дають змогу поглибленому розумінню сфери нашого дослідження.

Критерії відбору полягали в тому, що компанії застосовували гнучкі методи протягом останніх трьох років і отримали перший досвід роботи з гнучкими методами. Іншим важливим критерієм було те, що тільки команди SD використовують гнучкі методи, тоді як інші команди використовують методи, керовані планом

Чотири компанії, придбані у вибірці, розташовані в одному географічному регіоні. Однак фірми відрізняються за розміром, галуззю та досвідом роботи з гнучкими методами. Проектні групи наших кейс-компаній використовують Scrum або Kanban як гнучкі методи SD, а модель водоспаду ― як керований планом метод (див. таблицю 2.1).

Серед проектів, які було проаналізовано, приблизно від 50% до 70% залучених членів команди використовували гнучкі методи. Використано в данному дослідженні теоретичну вибірку до того моменту, коли було досягнуто теоретичного насичення, щоб зменшити зміщення вибірки та збільшити охоплення даними.

Таблиця 2.1

Опис випадків

Справа	FinanceCo	CommerceCo	E-commerceCo	MediaCo
Заснований	1890 рік	1995 рік	1999 рік	1992 рік
Співробітники	140 000	800	1000	500
Agile SD метод	Scrum	Scrum	Kanban і Scrum	Kanban
Керований планом метод	водоспад	водоспад	водоспад	водоспад
Число команд	5 Scrum команд 2 IT-операції 1 бізнес-команда	3 Scrum команди 2 IT-операції 2 бізнес-команда	4 команди Kanban 3 Scrum команди 1 бізнес-команда	2 команди Kanban 1 бізнес-команда
Посада співрозмовника (Роки досвіду з Agile методами)	№ 1 PO (<1) № 2 SM (>3) № 3 IT-операції (<1) № 4 Розробник (>3) № 5 CPO (>2)	№ 1 PO (>1) № 2 IT-операції (>1) № 3 Розробник (>1) № 4 SM (>5) № 5 Менеджер (>1)	№ 1 тестувальник якості (>5) № 2 Менеджер. (>2) № 3 SRM (<1) № 4 Розробник (>1) № 5 SDM (>2)	№ 1 SDM (>2) № 2 Розробник (>1) № 3 Agile тренер (>2) № 4 Менеджер (<1) № 5 IT-менеджер (>2)
Примітка. PO = власник продукту, SM = майстер scrum, CPO = головний власник продукту, QA = забезпечення якості, SRM = менеджер запитів на обслуговування, SDM = менеджер з надання послуг.				

2.4. Збір та аналіз данних

Було проведено напівструктуровані особисте інтерв'ю на місці з 21 респондентом, які мали детальні знання про впровадження та застосування гнучких методів у SD. На основі методу обґрунтованої теорії (GTM) теоретична вибірка дозволила нам вибрати партнерів для інтерв'ю на основі аналізу раніше проведених

інтерв'ю. Співбесіди тривали від 45 до 70 хвилин. Запитання були відкритими, що дозволяло респондентам вільно передавати свій досвід, погляди та висловлювання соціально складних контекстів, які передбачають використання гнучких методів. Усі інтерв'ю були записані, анонімні та розшифровані. Дані були закодовані за допомогою програмного забезпечення NVivo 11. Також дотримувались рекомендацій щодо інтерв'ю, які були уточнені під час збору даних. Було опитано членів принаймні двох команд з кожної організації. Розпочато було з того, що ставили досить загальні запитання про досвід опитуваних та досвід роботи з плановими та гнучкими методами. Щоб уникнути упередженості дослідників, респондентіи надавали наставлення про особистий та командний досвід роботи з agile методами. Далі запитання були білшь зосередженні на співпраці між командами. Незабаром було зрозуміло, що між командами різних методів може виникнути напруга. Таким чином, виникла напруженість і відповідні рішення стали предметом наших інтерв'ю. З більшим розумінням командної співпраці та напруженості, що виникла, було вирішено звернутися до попередньої літератури та почали враховувати відповідні концепції. Оскільки GTM передбачає безперервну взаємодію між збором та аналізом даних, сукупність знань. Окрім інтерв'ю, дослідження включає внутрішні (наприклад, agile ради та внутрішні комунікації) та зовнішні (наприклад, річні звіти) джерела. Використання кількох джерел даних, тріангуляція даних дозволили підвищити внутрішню валідність та пом'якшити потенційні упередження.

Для аналізу даних використовулося інформовану GTM для виконання трьох раундів кодування. Також було обрано індуктивний дослідницький підхід для вивчення співіснування планових і гнучких методів у командах, які співпрацюють. Використано підхід до кодування Штрауса та почали з відкритого кодування для визначення початкових категорій. На другому кроці застосувано осьове кодування, щоб отримати більш глибокі знання та зв'язати категорії одна з одною. Використовуючи вибіркове кодування, було визначено основну категорію та пов'язані з нею інші категорії. Протягом цього процесу постійно порівнювалися дані, щоб забезпечити суворе кодування, і постійно зверталися до існуючої літератури та теорій, щоб підтвердити наші висновки або виявити можливі суперечливі пояснення.

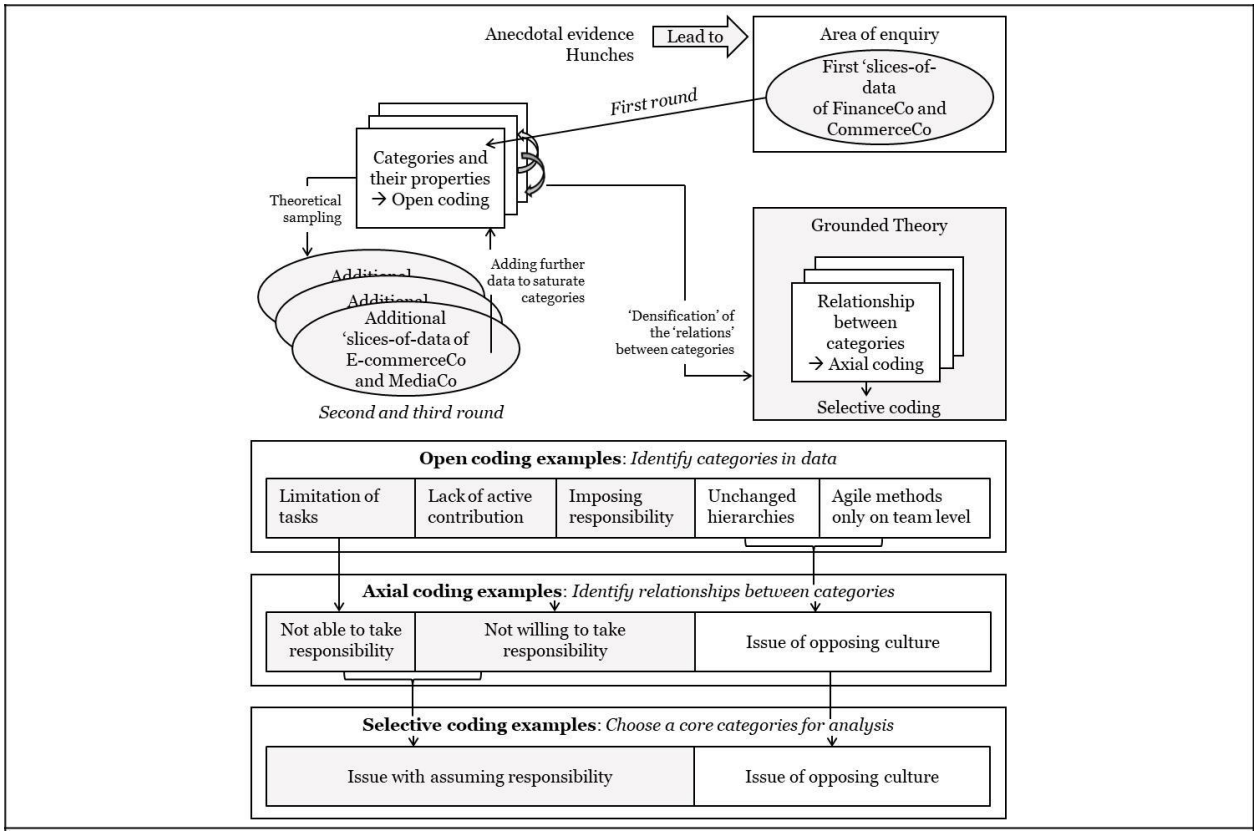

Рисунок 2.3 Цикл збору та аналізу даних

Перший етап, почав з невеликого набору інтерв'ю у FinanceCo та CommerceCo. Було відібрано організації, оскільки вони нещодавно застосували гнучкі методи. Компанії дозволили провести поглиблене дослідження команд, залучених до процесу розробки, оскільки ключові види діяльності та взаємодії були встановлені та відтворені. Співбесіди були відкритими та розглядали поточний стан усиновлення. Наше початкове відкрите кодування виявило адаптації на рівні команди в результаті прийняття методів, орієнтованих на планування, до гнучких.

Другий етап, початкове розуміння з першого раунду інтерв'ю було теоретично засноване на попередній літературі. Було вкористанно AST для подальшого аналізу та розуміння даних. Застосовуючи теорію, ми виявили напруженість, що виникає внаслідок прийняття гнучких методів. AST використовувався як пояснення наших висновків, але ми залишалися відкритими і дозволяли з'являтися подальшим концепціям протягом усього процесу. Також включено в дослідження E-commerceCo, яка використовувала Scrum і Kanban протягом трьох роківщоб збагатити дані та

проаналізувати пізнішу стадію процесу прийняття. Після порівняння інцидентів з міжорганізаційними даними, щоб визначити відмінності в подіях, діяльності та процесах, виявлено схожі виникаючі напруження та стратегії вирішення. На цьому етапі виявилося, що AST відповідає нашим проведеним даним.

Останній етап, було розширено вибірку, включивши інтерв'юйованих на керівні посади, щоб досягти більш чіткого розуміння співпраці між гнучкими та керованими планами командами. Також інтегрувано четверту компанію, яка використовує Kanban, до вибірки, щоб виключити напруженість, пов'язану з методами, та узагальнити наші висновки. Рисунок 3 підсумовує процес збору та аналізу даних і вказує зразкові коди для ілюстрації процесу кодування.

2.5. Результатативний аналіз

2.5.1. Внутрішній аналіз

FinanceCo є міжнародною фінансовою та страховою компанією, яка запровадила Scrum крок за кроком для підвищення прозорості процесів для всіх зацікавлених сторін. Перш ніж використовувати гнучкі методи, усі команди, залучені до процесу розробки, використовували підхід каскаду без будь-яких циклів зворотного зв'язку. Організаційна структура залишилася незмінною завдяки застосуванню гнучких методів, а це означає, що команди, які керують клієнтськими платформами компанії, залишалися розділеними на бізнес-коианди та IT-команди, які включали команди SD та IT-операцій.

Перша agile-команда складалася з власника продукту (PO), Scrum-майстра (SM) і чотирьох розробників. З точки зору дисциплінарної відповідальності, члени IT-команди (тобто розробники або SM) звітували перед керівником команди SD, тоді як PO підпорядковується керівнику бізнес-команди. З точки зору функціональної відповідальності, гнучка команда підпорядковується керівництву бізнес-команди, оскільки гнучка команда є частиною бізнес-функції. Почавши з однієї гнучкої команди, FinanceCo за два роки збільшила Scrum до п'яти команд SD. Щоб

координувати всі спритні команди, а менеджер проекту проводить збори всіх scrum-майстрів, щоб контролювати прогрес своїх команд (SM). CPO координує ОП та звітує перед керівництвом. Після використання Scrum протягом трьох років, FinanceCo об'єднала IT та бізнес-основи в один і почала використовувати підхід DevOps.

CommerceC — це телеторгова компанія, що працює в регіоні EMEA з фокусом на продажі фізичних товарів. Щоб йти в ногу зі швидкістю своїх природжених цифрових конкурентів і збільшити продажі на своїй онлайн-платформі, CommerceCo представила Scrum в SD. Іншою причиною прийняти гнучкі методи було те, що «ринок праці, пропонує переважно розробників з гнучкими профілями навичок, які не бажають використовувати метод водоспаду для розвитку на існуючій платформі".

Структура компанії складається з трьох відділів з чотирма командами SD. Впровадження гнучких методів почалося з однієї команди SD, яка була поширена на всі чотири команди за один рік. Дві команди IT-операцій продовжували використовувати метод водоспаду. Agile-команди складалися з PO, SM та трьох-п'яти розробників. Усі Agile PO є частиною бізнес-команди, яка організована на основі проекту з використанням моделі водоспаду. Щоб забезпечити безперервний досвід роботи з клієнтами, менеджери процесів підтримують гнучкі команди на технічному рівні. CPO координує роботу всіх Scrum-команд і виступає в якості ключа між відділами та керівництвом. Спільноти практичних зустрічей пропонують розробникам можливість обмінятися ідеями на рівні домену (наприклад, дизайн інтерфейсу). Розглянувши можливість інтеграції операційних команд SD та IT.

E-commerceCo керує національною платформою електронної комерції, структурованою за бізнес-одиницями відповідно до шести продуктових доменів. Після використання моделі водоспаду протягом багатьох років одна команда прийняла Scrum, а протягом трьох років усі інші команди пішли за ним. Спочатку всі agile-команди використовували Scrum, але з часом чотири команди прийняли Kanban, оскільки здавалося більш доцільним забезпечити безперервну доставку за межами спринту. В результаті чотири команди використовували Kanban, а три – Scrum.

Команда Scrum або Kanban складається з представника продукту і чотирьох-семи розробників. SM приєднується до команд Scrum лише за запитом, тоді як у

Kanban-команд немає менеджера з надання послуг (SDM). Для дотримання платформного підходу холдинг об’єднує всі продукти, розроблені дочірніми бізнес-підрозділами. Платформний підхід мінімізує необхідний обмін між розробниками. ОП щотижня зустрічаються з керівником відповідного домену. Agile-команди в межах домену звітують до керівник групи, який не залежить від їхніх завдань, а технічний керівник контролює всіх розробників. E-commerceCo дотримується підходу DevOps, де «команди відповідають за підтримку розробленого програмного забезпечення».

MediaCo є національним мовником, який прийняв Kanban два роки тому, щоб конкурувати з ринком відео на вимогу, що швидко змінюється. Раніше MediaCo тривалий час використовував модель водоспаду з детальними специфікаціями вимог і плануванням проекту. Компанія поділяється на бізнес та IT-відділ. Після прийняття структура залишилася незмінною. Однак обидві команди SD, що знаходяться в IT-відділі, прийняли Kanban і сформували гнучкі команди. SDM підтримує кожну гнучку команду. Команда Kanban складається з SDM та шести-восьми розробників. Через обмежені ресурси MediaCo використовує SDM як SRM для посередництва між бізнес-командами та командами, що сприяють адаптації. Фактичний SRM знаходиться в маркетинговій команді, яка ставить вимоги. MediaCo дотримується підходу DevOps, що означає, що команди Kanban розгортають і підтримують розроблене програмне забезпечення. З постійним використанням канбан, гнучких методів все більше звертаються до різних команд і поширюються за межі SD.

2.5.2 Перехрестний аналіз

Далі було представлено результати перехресного аналізу, проілюстрованого на малюнку 4, починаючи з фази введення з опису джерел структури та внутрішньої системи наших компаній. У фокусі аналізу є етап процесу. На цьому етапі було описано напруженість, що виникає внаслідок співіснування планових і гнучких методів, і детально розбирано, як ці напруження вирішувалися за допомогою нових джерел структури.

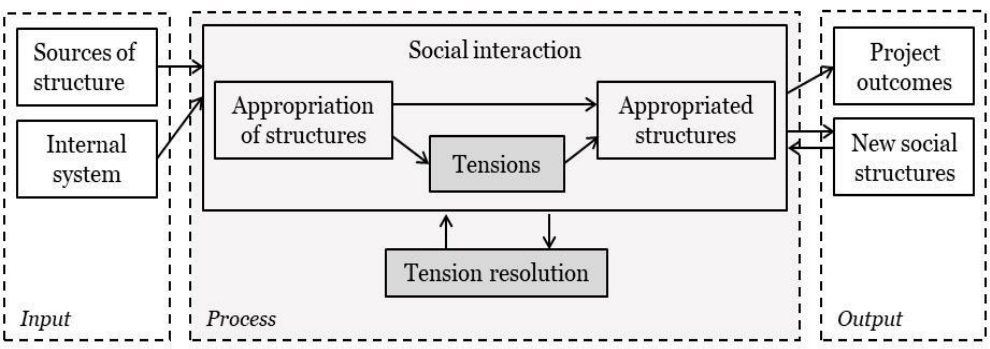

Рис. 2.4 Напруга, що виникає на етапі процесу.

Фаза введення включає в себе керовані планом і гнучкі методи як джерела структури та внутрішньої системи. Перед прийняттям усі організації застосовували модель водоспаду для SD включаючи тривалий процес вимог і важку документацію. Таким чином, структурні особливості планових методів були подібними у всіх чотирьох випадках, включаючи ієрархічну структуру для здійснення дисциплінарного контролю та послідовний план дотримання суворої політики та процедур.

FinanceCo, CommerceCo та E-commerceCo спочатку впровадили Scrum відповідно до рекомендацій. MediaCo представила Kanban також відповідно до вказівок методу: «Ми почали з дотримання принципу «почніть з того, що ви робите зараз», і ввели обмеження на незавершену роботу. Деякі команди E-commerceCo прийняли Kanban, визнаючи рекомендації методу. Канбан вважався більш підходящим для забезпечення безперервної доставки поза межами спринту. Таким чином, всі компанії запровадили agile методи відповідно до agile метод структури.

Що до спритного духу, всі організації сприяють культурі невдач із прозорими процесами та відкритим зворотним зв’язком. Усі команди відкриті до гнучких методів і орієнтуються на клієнта, зосереджуючись на комунікації та ітераційних процесах. Щодо внутрішньої система, організації покладаються на командно-адміністративне керівництво і не мали або мали небагато досвіду роботи з гнучкими методами заздалегідь. Жодна з чотирьох компаній не провела повного навчання. Наприклад, працівникам CommerceCo та E-commerceCo було надано лише один

навчальний день. Пізніше в процесі, FinanceCo і CommerceCo найняли розробників з досвідом у сфері agile, щоб сприяти впровадженню гнучких методів.

Що стосується фази процесу, усі компанії, що займаються справами, описали, що вони використовували метод водоспаду протягом десятиліть. Окрім привласнення структур, керованих планом, відповідні agile методи були призначені відповідно до загальних рекомендацій у всіх компаніях.

Усі кейс-компанії впровадили гнучкі методи для відображення проектів SD привласнення рухається. Окрім гнучких команд, команди бізнес- та IT-операцій продовжували використовувати планові методи та цінні принципи водоспаду (наприклад, детальне планування та контроль проекту).

Мета привласнення організацій (інструментальне використання) полягало в тому, щоб використовувати переваги гнучких методів, протистояти тиску конкуренції та відповідати на мінливі потреби споживачів. Загальнийставленняпо відношенню до agile методів було в основному позитивним. Організація проекту, керована планом, розчарувала більшість співробітників і змусила їх вітати прийняття.

2.6. Загальний аналіз

Після привласнення обох джерел структури виникла напруженість через співіснування методів, керованих планом, і які вирішувалися за допомогою нових джерел структури.

По-перше, співіснування методів викликало напругу щодо бюджетування. Усі компанії продовжували розподіляти бюджети традиційним способом (наприклад, визначаючи попередні бюджети для всіх проектів). Необхідність бюджетного контролю спричинила напругу між бізнес-командою, яка наглядає за проектом, і гнучкими командами. Бізнес-команда, яка керується планами, використовувала для визначення планів проекту з регулярними звітами про стан зустрічей. Після кожної ітерації керівництво проекту вказувало бюджет, витрачений на розробку. Однак процес переоцінки продукту на початку кожної ітерації відповідно до бюджету, що

залишився, був новим для бізнес-командивикликав страх перед перевищенням бюджетів.

Таким чином, планований бюджетний контроль, який здійснював бізнес-команда, гальмував процес розробки та гальмував нові проекти. Лише E-commerceCo уникла напруженості щодо бюджету, організувавши компанію на продуктові підрозділи та розподіливши бюджети окремо для кожного підрозділу. Іншим компаніям довелося шукати інші шляхи вирішення напруженості щодо бюджету. MediaCo почала експериментувати з більш потужним і адаптивним способом управління, ніж традиційний командно-контрольний. Однак компанія все ще дуже покладається на бюджетний контроль, який виконує бізнес-команда, оскільки розподіл бюджетів, особливо для нових проектів, залишається проблемою.

По-друге, напруженість у знаннях виникла через те, що кілька співробітників різних відділів утворюють спритну команду. Перед тим, як приєднатися до agile-команди, деякі члени команди виконували завдання, які охоплювали лише частину процесу розробки. Таким чином, прийняття agile методів спричинило зміни в структурі команди, які торкнулися членів agile-команди через брак інформації, необхідної для повсякденної роботи.

Подібно до E-commerceCo, PO CommerceCo було призначено для нагляду за всіма характеристиками продукту і, отже, не змогло забезпечити відповідність попереднім функціям і знати взаємозалежність усіх характеристик продукту. ПП не в змозі призначати завдання, що відповідають процесу. Крім того, справа MediaCo виявила напруженість знань для SRM. У FinanceCo напруженість знань виникла внаслідок включення розробки вимог та збору вимог клієнтів. З дуже складним продуктом розробки одна роль навряд чи могла б об’єднати необхідні знання для автономної обробки всіх запитів. Напруга в знаннях була вирішена двома способами нашими кейс-компаніями. CommerceCo, наприклад, створила нову команду під назвою менеджери процесів, яка забезпечувала безперервне обслуговування клієнтів.

Крім того, напруженість процесу виникла через втручання команд, керованих планом, у гнучкі процеси. Швидкі безперервні процеси є ключовими особливо для гнучких команд, щоб керувати своїми короткими ітеративними робочими процесами

без перерв. Порушення гнучких робочих процесів спричинить порушення процесу, які неможливо відкликати. Передавши програмне забезпечення команді IT-операцій для обслуговування, цей робочий процес було перервано. На відміну від цього, бізнес-команда втручалася в процес, що змінюється, відмінюючи відповідальність PO або SRM щодо визначення пріоритетів завдань і здатність розробників виконувати визначену роботу.

Аналогічно в MediaCo бізнес-команда регулярно ігнорувала відповідальність SRM і включала функції, над якими повинна працювати гнучка команда, з найвищим пріоритетом. У підсумку, співпраця між бізнес-командами та IT-операційними командами викликає напруженість процесів у FinanceCo, E-commerceCo та MediaCo. Будучи в перший рік застосування методів Agile, CommerceCo організувала процеси, як і раніше, за планом і не відчувала напруги в процесі. Надання можливості гнучкої команді самостійно об'єднати своє програмне забезпечення в головну гілку вирішило величезну перешкоду. Можливість об'єднати код без його перевірки IT-операціями робить команду менш залежною від команди IT-операцій

По-третє, прийняття гнучких методів викликало напруженість у плануванні. Використання моделі водоспаду передбачало детальне планування етапів проекту бізнес-командами. Бізнес-команда, яка отримувала вимоги клієнтів, відповідала за планування збільшення продукту на заздалегідь визначену дату запуску та за надання цієї інформації клієнтам. Прийняття гнучких методів змінило цей підхід до детального планування. Agile-команди ітеративно переглядали обсяг розробки, щоб підвищити прозорість і забезпечити цінний випуск програмного забезпечення для замовника після кожної ітерації відповідно до фіксованих параметрів.

Непередбачуваність викликала занепокоєння щодо втрати контролю над обсягом розробки програмного забезпечення з боку бізнес-команд. Таким чином, спритні команди розвивалися автономно, їх контролювали бізнес-команди, які вважали, що гнучкі команди втратили планування та втратили з поля зору сферу розробки. Вони упускають сенс бути гнучкими – розробляючи продукт із найвищою можливою цінністю для споживача.

Напруженість у плануванні виникла в усіх компаніях, які вирішувалися, зробивши детальні дорожні карти обов'язковими для гнучких команд. Тому жодна з наших розглянутих компаній не знайшла способу розв'язати напруженість у плануванні, але всі вони дотримувалися керованих планом дорожніх карт. Однак у FinanceCo та MediaCo ми знайшли перші підходи для спрощення необхідної інформації дорожньої карти, щоб зменшити робоче навантаження гнучких команд і підвищити гнучкість.

Іншою напругою, була напруга відповідальності. Гнучкі методи передбачають, що команди наділяються повноваженнями для забезпечення автономного наскрізного процесу розробки. Однак, оскільки лише команди SD застосовували гнучкі методи, команди, що не є гнучкими, ділові та IT-операційні команди залишалися відповідальними за частини процесу SD і перешкоджали наскрізній відповідальності гнучкої команди.

Аналіз показав, що ненадійні команди сповільнювали процес розробки, оскільки кілька обов'язкових процесів затвердження входили до їхньої відповідальності. Аналогічно, IT-операційна команда FinanceCo перешкоджала гнучким командам інтегрувати своє програмне забезпечення в систему без перевірки його членом команди IT-операцій. Окрім розподілених обов'язків між командами, гнучкі команди відмовляються брати на себе відповідальність у співпраці з не-спритними командами. Таким чином, наші справи показали два види напруженості відповідальності.

Так сама по собі напруженість, спричинена відсутністю відповідальності, наданої гнучким командам, а також, напругою, що виникає через відсутність відповідальності, яку беруть на себе agile-команди. Щоб розв'язати напруженість, що виникла у сфері відповідальності, FinanceCo здійснила масштабну організаційну реструктуризацію, об'єднавши команди бізнесу та IT. Реорганізувавши команди та вирівнявши ієрархію, співробітники виробили спільне розуміння та взяли на себе відповідальність за свої дії замість того, щоб передавати її іншим. Реорганізація в напрямку міждисциплінарних команд сприяла розвитку гнучкої культури та допомогла компанії далі використовувати гнучкі методи. У CommerceCo

напруженість вирішити не вдалося. Усі спроби, як-от запрошення персоналу з IT-операцій на ретроспективні зустрічі, поки що не увінчалися успіхом, оскільки команда IT-операцій відмовляється адаптуватися до гнучких методів.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому дослідженні розглядаються теоретичні та емпіричні прогалини в літературі щодо сумісних методів. Результати дослідження уточнюють теорію, щоб пояснити прийняття організацією рухливих команд у середовищах, керованих планами, що призводить до співіснування методів, керованих гнучкістю та планом. Це дослідження вносить свій внесок про agile методи двома способами.

Дослідження виявляє напруженість, яка виникає внаслідок співпраці команд, які керуються планом, і спритних команд. Спираючись на AST, ми досліджуємо співіснування методів, як соціальну взаємодію між спритними та не-спритними командами та виявляємо напругу між привласненням структур і впливом на привласнені структури. Тут показано, що напруженість вирішується за допомогою нових джерел структур, таких як розв’язання напруженості знань шляхом організації координаційної зустрічі. таким чином, ми розширюємо знання про виникнення нових джерел структури. Крім того, виявилено, що нові джерела структури або прискорюють, або уповільнюють прийняття гнучких методів. Це очевидно у випадку CommerceCo, де використовуються в основному збалансовані підходи до вирішення, які гальмують просування гнучких методів.

Окрім виявлення напруженості, що виникає внаслідок співіснування методів agile і планових методів, ми показуємо, що прийняття agile методів є багатошаровим явищем, яке тягне за собою реорганізацію структур, ролей і процесів. Дослідження показує, що забезпечення тісної співпраці між командами неминуче для досягнення повного переходу. Таким чином, після розв’язання напруженості, що виникає внаслідок поганої співпраці між гнучкими та керованими планами командами, організації змушені знайти більш інтегровані, міжфункціональні підходи, такі як

DevOps. Організація використовує DevOps як спосіб розширити гнучкі методи та зруйнувати функціональні розрізнення.

Також тут було запропонувано розширення AST шляхом включення напруженості на основі доказів чотирьох тематичних досліджень з організаціями, які впроваджують гнучкі методи в SD після використання методів, керованих планом, протягом десятиліть. Розглядаючи процес прийняття, ми можемо спостерігати співіснування методів лише для одного моменту часу.

На етапі збору та аналізу даних ми забезпечили надійність та внутрішню валідність наших даних. Крім того, узагальнюючи висновки на інші контексти, які включають напруженість, що виникає внаслідок співпраці, слід робити обережно, оскільки висновки можуть бути специфічними для використання конкретних напруженість від планових і гнучких методів, а також до специфічних для фірми факторів. Тим не менш, узагальнення результатів вимагає кількісних підходів, які перевіряють нашу адаптовану модель ASTПодальші дослідження можуть використовувати конкуруючі теоретичні лінзи, окрім AST, що може привести до нових додаткових уявлень щодо вирішення напруженості, що виникає внаслідок прийняття гнучких методів.

РОЗДІЛ 3
ЯКІСТЬ ПРОГРАМНИХ ПРОДУКТІВ

3.1. Модель ISO 25010

ISO 25010 є міжнародним стандартом для оцінки якості програмного забезпечення та систем. Цей стандарт пережив три важливі оновлення у 2007, 2011 та 2017 роках. Цей стандарт також відомий як модель SQuaRE (Вимоги та оцінка якості систем і програмного забезпечення). Він також описує якість програмного продукту та якість у використанні. Відповідно до ISO 25010 було розроблено на основі оновлення моделі ISO 9126. Згідно з ними, попередня модель (ISO 9126) має шість факторів і двадцять один підфактор. За допомогою простого порівняння двох моделей, «безпека» та «сумісність» були єдиними двома факторами, введеними разом із їхніми підфакторами в ISO 25010. Атрибути якості в цій моделі представлені, починаючи від верхніх факторів до підфакторів. Верхній рівень складається з восьми факторів, які далі розкладаються на тридцять один підфактор на нижньому рівні. ISO 25010, похідний від моделі ISO 9126, описує тридцять один атрибут, який повинен мати кожен якісний програмний продукт.

Багато дослідників адаптували стандарт ISO 25010, щоб запропонувати нові моделі якості у своїх дослідженнях. Як міжнародний стандарт, модель ISO 25010 також була адаптована в цьому дослідженні для розробки нової моделі якості ERP-систем.

3.1.1. Нова модель якості системи ERP

Незважаючи на те, що існують спеціальні моделі якості програмного забезпечення, розроблені для оцінки конкретних програмних продуктів, більшість моделей якості програмного забезпечення є загальними та загальними для всіх типів програмних продуктів. Наприклад, ISO 25010 містить фактори, які є загальними для оцінки якості кожного типу системи та програмних продуктів. Більше того, багато систем і програмних продуктів мають власні фактори або особливості, які необхідно враховувати під час оцінки. Тому для оцінки програмного забезпечення та систем продуктів, існуючі моделі якості програмного забезпечення повинні бути ретельно відібрані, модифіковані або розширені. Це означає, що коефіцієнти якості та субфактори моделі якості мають бути скориговані, щоб вони відповідали новій системі, що оцінюється, а не навпаки. Таким чином, це дослідження адаптує ISO 25010 для оцінки якості ERP-систем у ВНЗ.

Хоча є декілька досліджень якості програмних продуктів на основі моделі ISO 9126 в освітніх середовищах дослідження щодо адаптації моделі ISO 25010 для оцінки ERP-систем у ВНЗ є дуже рідкісними. Таким чином, новинкою цього дослідження є запропонована модель якості на основі ISO 25010 для оцінки якості ERP-систем у ВНЗ. Незважаючи на те, що ISO 9126 використовувався кілька разів при розробці інших моделей для оцінки систем ERP, ця нова модель ISO 25010 також була розроблена для її покращення, заміни та розширення. Оскільки системи та програмні продукти сьогодні стають дедалі складнішими та досконалими, для їх оцінки також необхідні нові моделі якості. Тому для оцінки ERP-систем у ВНЗ необхідні моделі якості на основі ISO 25010.

Багато дослідників адаптували ISO 25010 у своїх дослідженнях. Загальність моделі якості ISO 25010 дозволяє легко адаптувати її до розробки багатьох конкретних моделей якості програмного забезпечення, таких як модель якості системи ERP. Адаптуючи такі моделі, як ISO 25010, дослідники здебільшого усувають деякі атрибути або фактори якості, додають нові атрибути або перевизначають існуючі атрибути моделі. Це дослідження додає та перевизначає

субфактори стандарту ISO 25010, щоб відповідати моделі якості для оцінки систем ERP у ВНЗ. Нова модель якості системи ERP описує вісім факторів, включаючи функціональну придатність, надійність, зручність використання, ефективність продуктивності, сумісність, безпеку, ремонтопридатність і переносимість, які далі розкладаються на тридцять чотири підфактори. У цьому дослідженні в модель ISO 25010 було введено три нові субфактори. Підтримка та можливість пошуку були додані як підфактори зручності використання, а можливість архівації також була додана як підфактор безпеки. У наступному розділі описано три нові субфактори, існуючі фактори та підфактори моделі ISO 25010, адаптованої для цього дослідження.

Функціональна придатність

Цей коефіцієнт якості описує ступінь, до якого програмний продукт або система надає функції, які задовольняють заявлені та неявні потреби зацікавлених сторін при використанні за певних умов. Також його коефіцієнт якості поділено на три нижчі фактори, включаючи функціональну повноту, функціональну правильність та функціональну відповідність. Оскільки системи ERP у ВНЗ мають різноманітні функціональні додатки, має існувати такий фактор якості, як функціональна придатність, щоб допомогти оцінити ці функції. Таким чином, функціональна придатність була адаптована в новій моделі якості системи ERP.

Надійність

Коефіцієнт надійності виражає здатність системи або програмного продукту підтримувати свій рівень продуктивності або визначені функції за певних умов протягом певного періоду часу. Чотири нижчі фактори пов'язані з фактором надійності, а саме: зрілість, доступність, відмовостійкість і можливість відновлення. Цей коефіцієнт якості був адаптований у нашій новій моделі якості системи ERP для оцінки надійності різних функцій та послуг, які системи ERP надають у ВНЗ за певних зазначених умов.

Зручніть використання

Відповідно до коефіцієнт юзабіліті описує ступінь, до якої програмне забезпечення або системний продукт можуть бути використані для досягнення визначених цілей з ефективністю, ефективністю та задоволенням у певному контексті

використання. Коефіцієнт юзабіліті має набір нижчих факторів, які включають впізнаваність відповідності, придатність для навчання, працездатність, захист від помилок користувача, естетику інтерфейсу користувача та доступність. У цьому дослідженні підтримка та можливість пошуку були додані як нижчі фактори зручності використання, щоб допомогти оцінити роботу систем ERP у ВНЗ. Оскільки зручність використання є важливим фактором якості для кожного програмного забезпечення або систем, які включають системи ERP, його було адаптовано в нашій новій моделі якості системи ERP.

Ефективність продуктивності

Коефіцієнт ефективності продуктивності описує здатність програмного продукту або системи керувати заданою кількістю ресурсів для забезпечення та максимізації продуктивності. Цей коефіцієнт якості також був розбитий на три нижчі фактори, включаючи поведінка в часі, використання ресурсів і потужність. Ефективність роботи була адаптована до нової моделі якості системи ERP для оцінки розподілу ресурсів та використання систем ERP під час надання необхідних послуг та функцій у ВНЗ.

Сумісність

Фактор сумісності — це здатність програмних продуктів або системи взаємодіяти з іншими програмними продуктами чи системами без будь-яких збоїв. Тобто система ERP виконує свої необхідні функції, одночасно спільне використання того ж апаратного чи програмного середовища з іншими системами. Коефіцієнт сумісності має два нижчі фактори, а саме співіснування та сумісність. Знову ж таки, адаптація цього коефіцієнта якості в новій моделі якості системи ERP сприятиме оцінці обміну інформацією та спільного використання загального середовища системою ERP з іншими програмними продуктами та системами.

Безпека

Відповідно до фактору безпеки — це те, як програмні продукти чи системи захищають свою інформацію та дані (інформаційні ресурси) від неавторизованих осіб або від інших програмних продуктів чи систем. Коефіцієнт безпеки має низку нижчих факторів, які включають конфіденційність, цілісність, невідмовність, підзвітність та автентичність. У цьому дослідженні архівованість була додана як нижчий фактор безпеки для оцінки роботи ERP-систем у ВНЗ. Оскільки безпека є важливим фактором якості для кожного програмного забезпечення або систем, які включають системи ERP, його було адаптовано до нашої нової моделі якості системи ERP.

Ремонтопридатність

Здатність програмних продуктів або систем бути модифікованими, виправленими або адаптованими до поточних змін у середовищі описує їх особливість ремонтопридатності. П'ять нижчих факторів, включаючи модульність, можливість повторного використання, аналізованість, модифікованість і тестованість, були пов'язані з ремонтопридатністю. Застосування цього коефіцієнта якості до нової моделі якості системи ERP приведе до того, що системи ERP у ВНЗ повинні дозволяти модифікації або виправлення без особливих труднощів.

Портативність

Можливість перенесення програмних продуктів або систем з одного апаратного, програмного або іншого операційного або використовуваного середовища на іншу операційну платформу визначає їх характеристику переносимості. Три нижчі фактори, включаючи адаптивність, можливість встановлення та замінність, описують функцію портативності. Цей коефіцієнт якості був адаптований до нової моделі якості системи ERP для оцінки роботи систем ERP на різних апаратних і програмних платформах і в різних середовищах. У світлі всього аналізу, проведеного в цьому дослідженні, у (табл. 3.1) та на (рис. 3.1) представлена нова модель якості, заснована на стандарті ISO 25010. Ця нова модель якості включає вісім основних факторів і тридцять чотири субфактори. Запропонована модель показує, як ці фактори та підфактори якості були адаптовані для оцінки якості ERP-систем у ВНЗ.

Таблиця 3.1

Модель якості системи ERP у ВНЗ

Фактор	Суб-фактор	Пояснення
Функціональна придатність	Функціональна завершеність	Чи покриває система ERP усі визначені завдання та цілі користувача?
	Функціональна коректність	Чи може система ERP забезпечити правильні результати з необхідним ступенем точності?
	Функціональна відповідність	Чи сприяє функція ERP-системи виконання визначених завдань і цілей?
Надійність	Зрілість	Чи відповідає система ERP потребам своїх користувачів під час нормальної роботи?
	Доступність	Чи може система ERP бути оперативною та доступною в потрібний для використання час?
	Відмовостійкість	Чи може система ERP працювати за планом, незважаючи на наявність несправності апаратного або програмного забезпечення?
	Відновлюваність	Чи може ERP-система під час катастрофи відновити її та відновити її до бажаного стану?
Зручність використання	Доречність впізнаваність	Чи може система ERP бути легко визнана користувачами як відповідний продукт або система для вирішення їхніх потреб?
	Можливість навчання	Чи можна легше вивчити систему ERP?
	Працездатність	Чи можна легко керувати та керувати системою ERP?
	Захист від помилок користувача	Чи захищає система ERP користувачів від помилок?
	Естетика інтерфейсу користувача	Чи виглядає інтерфейс користувача системи ERP приємним і задовільним?
	Доступність	Чи можуть певні користувачі отримати доступ до системи ERP за певних умов?
	Підтримка	Чи може система ERP надавати базові інструкції та підказки своїм користувачам під час роботи?

Продовження таблиці 3.1

Ефективність продуктивності	Поведінка в часі,	Чи може система ERP реагувати та обробляти події швидше?
	Використання ресурсів	Чи може система ERP ефективно використовувати інформаційні ресурси?
	Ємність	Чи відповідають параметри системи ERP їхнім системним вимогам?
Сумісність	Співіснування	Чи може система ERP ефективно виконувати необхідні операції, ділячись своїм середовищем та інформаційними ресурсами з іншими продуктами чи системами?
	Сумісність	Чи може система ERP взаємодіяти з іншими системами або програмними продуктами?
Безпека	Конфіденційність	Чи може система ERP забезпечити доступ до інформаційних ресурсів лише тим, хто має право на доступ?
	Цілісність	Чи запобігає система ERP несанкціонований доступ до інформаційних ресурсів або зміна даних?
	Не відмова	Чи може система ERP довести дію або подію, які, як вважають, мали місце?
	Підзвітність	Чи може система ERP однозначно відстежити або врахувати дію чи подію суб’єкта?
	Автентичність	Чи можна використовувати систему ERP для ідентифікації користувачів і ресурсів?
	Архівність	Чи зберігає і захищає система ERP свої минулі записи для подальшого використання?
Ремонтопридатність	Модульність	Чи складається система ERP з окремих компонентів або модулів для легкого використання?
	Можливість багаторазового використання	Чи можна використовувати модулі в системі ERP для роботи з іншими модулями в тій самій системі ERP?

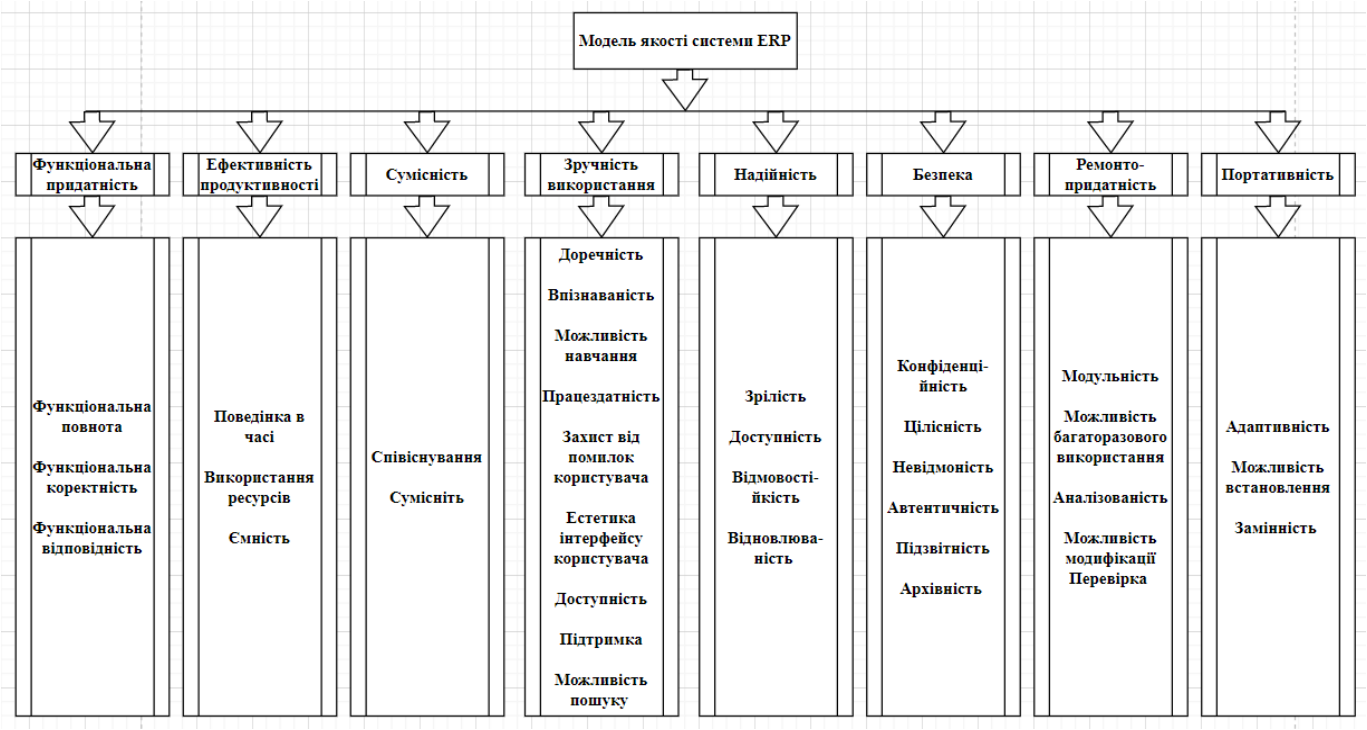

Рис. 3.1 Модель якості ERP

3.2. Область застосування

Цей стандарт визначає:

Модель якості при використанні, до складу якої входять п'ять характеристик, деякі з яких, у свою чергу, поділені на підхарактеристику. Ці характеристики стосуються результату взаємодії при використанні продукту в певних умовах. Дана модель застосовна при використанні повних людино-машинних систем, включаючи як обчислювальні системи, так і програмні продукти;

Модель якості продукту, до складу якої входять вісім характеристик, які, у свою чергу редь. підрозділяються на підхарактеристики. Характеристики відносяться до статичних та динамічних властивостей програмного забезпечення та обчислювальних систем. Модель застосовна як до комп'ютерних систем, так і до програмних продуктів.

Характеристики, що визначаються обома моделями, застосовні до будь-яких програмних продуктів і комп'ютерним системам. Характеристики та підхарактеристики забезпечують єдину термінологію для визначення специфікації,

вимірювання та оцінки якості систем та програмного забезпечення. Моделі надають безліч характеристик якості, з якими для повноти картини можна порівняти заявлені вимоги до якості.

Область застосування моделей не включає в себе чисто функціональні властивості, однак у неї включена функціональна придатність.Область застосування моделей якості включає специфікацію підтримки та оцінку програмного забезпечення і переважно програмних обчислювальних систем з різних точок зору. які пов'язані з їх придбанням, вимогами, розробкою, використанням, оцінкою, підтримкою, обслуговуванням, забезпеченням якості та управлінням ним. а також менеджментом та ау дитом. Моделі можуть, наприклад, використовуватися розробниками, набувачами, персоналом забезпечення якості та управління ним, а також незалежними оцінювачами, особливо відповідальними за специфікацію та оцінку якості програмного продукту. Діяльність під час розробки продукції, при якій можуть бути використані моделі якості, включає в себе;

- визначення вимог до програмного забезпечення та системи;
- підтвердження повноти визначення вимог;
- визначення цілей проектування програмного забезпечення та системи; визначення цілей тестування програмного забезпечення та системи;
- ідентифікацію критеріїв контролю якості в рамках забезпечення якості;
- визначення критеріїв приймання програмного продукту або переважно програмної обчислювальної системи;
- встановлення необхідних цього показників характеристик якості.

Відповідність

Будь-яка вимога до якості, специфікація якості або оцінка якості відповідають справжньому стандарту тільки в тих випадках, якщо: використовуються моделі якості або використовується адаптована модель якості, всі зміни якої обґрунтовані і для кото рій забезпечується відображення стандартної моделі.

Якість системи — це ступінь задоволення системою заявлених і потреб різних зацікавлених сторін, що дозволяє, таким чином, оцінити переваги. Ці заявлені та потреби представлені в міжнародних стандартах серії SQuaRE за допомогою моделей якості, які представляють якість продукту у вигляді розбивки на класи характеристик, які в окремих випадках далі поділяються на підхарактеристики. Подібна ієрархічна декомпозиція забезпечує зручну розбивку якості продукту на класи. Однак безліч під характеристик, пов'язаних з характеристикою, обраною для представлення типових проблем, необов'язково буде вичерпним.

Виміряні, пов'язані з якістю властивості системи називають властивостями якості, пов'язаними з відповідними показниками якості. Щоб прийти до показників характеристики або підхарактеристики якості у випадках, коли характеристика або підхарактеристика не може бути безпосередньо виміряна, необхідно ідентифікувати підмножина властивостей, яка в сукупності покриває характеристику або підхарактеристику, отримати показники якості для кожної властивості. об'єднавши їх у обчислювальному відношенні, досягти отриманого показника якості, що відповідає характеристиці або підхарактеристиці якості. На (рис. 3.2) показані відносини між характеристиками та підхарактеристиками якості та властивостями якості.

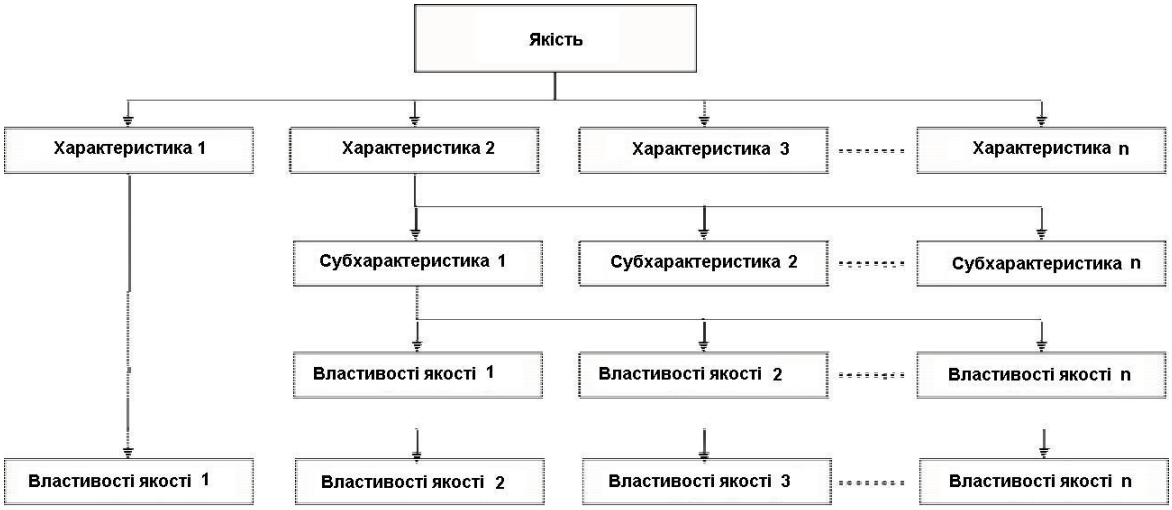

Рис. 3.2 Структура, що використовується для моделей якості

В даний час у серії SQuaRE є три моделі якості: модель якості при використанні і модель якості продукту, визначені в цьому стандарті, і модель якості даних. Спільне використання моделей якості дає основу.вважати, що враховані всі характеристики якості. Дані моделі забезпечують безліч характеристик якості, в яких зацікавлене широке коло осіб, таких як: розробники програмного забезпечення, системні інтегратори, набувачі, власники, спеціалісти з обслуговування, підрядники, професіонали забезпечення та управління якістю та користувачі.

Не всі характеристики якості з повної множини, що забезпечується цими моделями, є значимими для конкретної зацікавленої сторони. Проте кожна категорія заінтересованих осіб повинна бути врахована при аналізі та розгляді важливості характеристик якості для кожної моделі до завершення формування набору характеристик якості, які будуть використовуватися, щоб встановити, наприклад, вимоги до продуктивності продукції та системи або критерії оцінки.

3.2.1. Модель якості при її використанні

Модель якості при використанні визначає п'ять характеристик, пов'язаних з результатами взаємодії з системою: результативність, продуктивність, задоволеність, свободу від ризику та покриття контексту. Кожна характеристика застосовна до різних видів діяльності зацікавлених осіб, наприклад, для взаємодії оператора чи підтримки розробника.

Рис. 3.3 Якість у використанні моделі

Якість у використанні системи характеризує вплив, який має продукт (система або програмний продукт) на зацікавлених сторін. Вона визначається якістю програмного забезпечення, обладнання та робочого середовища, і від характеристик користувачів, завдань і соціального середовища. Всі ці фактори сприяють підвищенню якості у використанні системи.

3.2.2. Модель якості продукту

Модель якості продукту зводить властивості якості системи/програмного продукту до восьми характеристик, якими є: функціональна придатність, рівень продуктивності. сумісність, зручність користування, надійність, захищеність, супроводжуємося переносимість. Кожна характеристика, у свою чергу, складається з ряду відповідних підхарактеристик. Модель якість продукції можуть бути застосовані до просто програмного продукту, або до комп'ютерної системи, яка включає в себе програмне забезпечення, так як більшість з субхарактеристик актуальні як для програмного забезпечення і систем.

3.2.3. Цілі моделей якості

Модель якості продукції спрямована на цільову комп'ютерну систему, яка включає цільовий програмний продукт ,і якість у використанні моделі фокусується на всій системі, що містить цільову комп'ютерну систему і цільовий програмний продукт. Комп'ютерна система також включає комп'ютерне обладнання, нецільові програмні продукти, нецільові дані і цільових дані, які є предметом моделі якості даних. Комп'ютерна система включена в інформаційну систему, яка також може включати в себе одну або більше комп'ютерних систем і систем зв'язку, такі як локальній мережі і Інтернет. Інформаційна система в складі поєднання людини з комп'ютерною системою може включати користувачів, технічну і фізичну середу використання, де межа системи залежить від обсягу вимог або оцінки, і від того, хто є користувачем.

Наприклад, якщо користувачі повітряного судна з системою управління польотом - то система, від якої вони залежать включає членів екіпажу, планер і апаратне і програмне забезпечення в системі управління польотом, а якщо льотний екіпаж приймаються рівними користувачами, то система, від якої вони залежать, складається тільки з планера та системи управління польотом.

Інші зацікавлені сторони, такі як розробники програмного забезпечення, системні інтегратори, набувачі, власники, супроводжуючі, підрядники, забезпечують якість та професіоналізм управління, також будуть пов'язані з якістю.

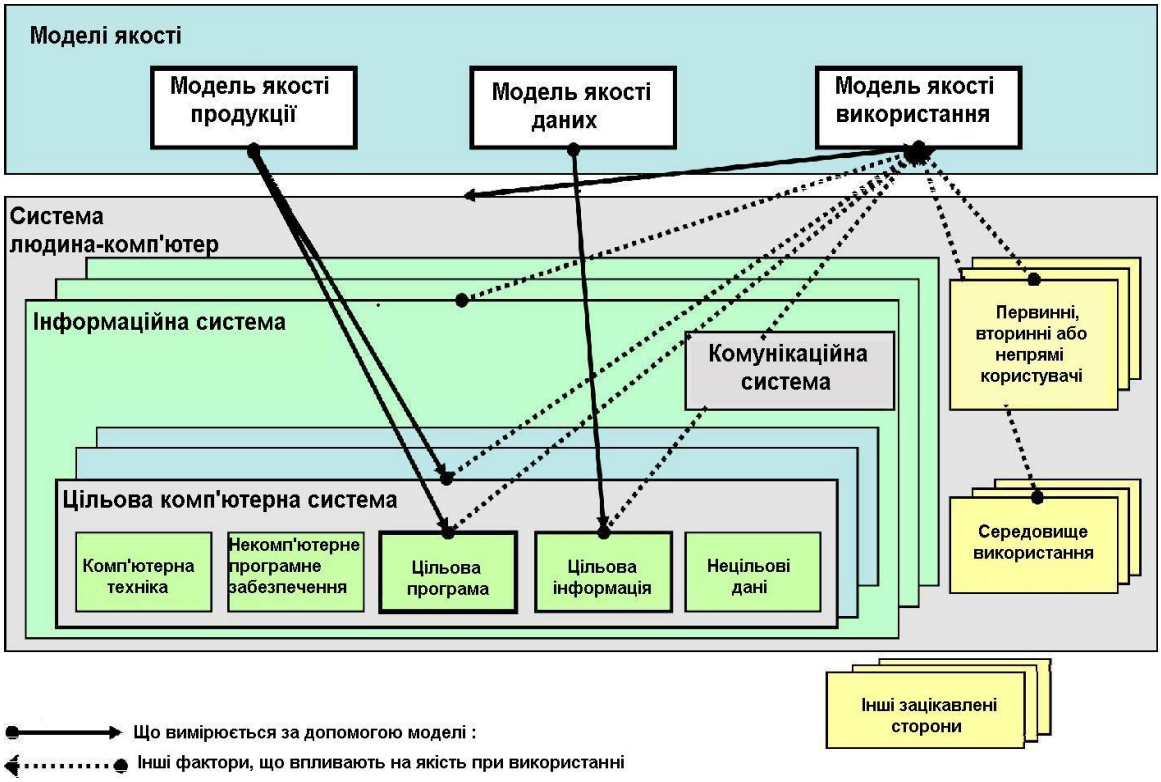

Рис. 3.4 Цілі моделей якості

3.2.4. Використання моделі якості

Якість продукції і якість в моделях використання корисні в визначенні вимог, розробці заходів, а також виконання оцінки якості. Певні якісні характеристики можуть бути використані в якості контрольного списку для забезпечення комплексного дотримання вимог до якості, забезпечуючи тим самим основу для оцінки, і як наслідок, зусиль і заходи, які будуть необхідні в процесі розробки систем.

Характеристики в якості моделі у використанні і якості продукції моделі призначені для використання в якості набору при визначенні чи оцінці комп'ютерної системи або якості продукту програмного забезпечення.

Це практично не можливо визначити чи виміряти всіх субзарактеристик для всіх частин великої комп'ютерної системи або програмного продукту. Аналогічно це зазвичай не практично для вказівок або вимірювання якості у використанні для всіх можливих поставлених користувачем завдань. Відносна важливість якісних характеристик буде залежати від цілей високого рівня і цілей проекту. Тому модель повинна бути адаптована перед використанням в якості частини розкладання вимог та визначити ті характеристики і субхарактеристики, які найбільш важливі, і кошти, що виділяються між різними типами вимірювань в залежності від цілей зацікавлених сторін і цілей продукту.

3.2.5. Якість з різних точок зору зацікавлених сторін

Моделі якості забезпечують основу для збору потреби зацікавлених сторін. Зацікавлені сторони включають такі види користувачів:

1. Основний користувач: людина, яка взаємодіє з системою для досягнення основних цілей.
2. Вторинні користувачі, які, наприклад, надають підтримку.
   a) контент-провайдер, системний адміністратор / адміністратор, менеджер з безпеки;
   b) супроводжуючий, аналізатор, портьє, інсталятор.

Таблиця 3.2

Приклади потреб користувачів для якості у використанні і якості продукції

Вимоги користувача	Первинний	Вторинні користувачі		Непрямий
		Контент-провайдер	Супроводжуючий	
	Взаємодія	Взаємодія	Підтримка	Вихід результату
Ефективність	Наскільки ефективно виконується завдання, поставлене користувачем?	Як ефективно має бути виконано завдання на виході?	Наскільки ефективна має бути підтримка та портування системи?	Як ефективно використовується завдання системи на виході?
Продуктивність	Як продуктивно виконується завдання, поставлене користувачем?	Як продуктивно задано потреби користувача до системи?	Як продуктивна має бути підтримка та портування системи?	Як продуктивно використовується завдання системи на виході?
Насичення	Як насичено виконується завдання, поставлене користувачем?	Як задоволені вимоги до системи задані користувачем?	Як задоволена має бути підтримка та портування системи?	Наскільки задоволено завдання системи на виході?
Свобода від ризику	З яким ризиком виконується завдання, поставлене користувачем?	З яким ризиком поставлено вимоги до системи користувачем?	З яким ризиком має бути підтримка та портування системи?	З яким ризиком являється завдання системи на виході?
Надійність	Як надійно виконується завдання, поставлене користувачем?	Які вимоги до надійності системи задані користувачем?	З якою надійністю має бути підтримка та портування системи?	З якою надійністю являється завдання системи на виході?
Безпека	Як безпечно виконується завдання, поставлене користувачем?	Які вимоги до безпеки системи задані користувачем?	Як безпечною має бути підтримка та портування системи?	Як безпечною являється завдання системи на виході?
Контекст охоплення	Як ефективною має бути система та вільною від ризику, щоб вільно використати всі потенційні контексти?	Як ефективним має бути контекст охоплення, щоб задовольнити контексти використання?	Як ефективним має бути контекст охоплення, щоб охопити всі потенційні контексти?	Як ефективним має бути контекст охоплення, щоб охопити всі потенційні контексти завдання на виході?

Продовження таблиці 3.2

Можливості розвитку	Як можна розвинути систему для доцільного використання всіх потенц. контекстів?	Як можна розвинути систему для доцільного використання всіх потенц. контекстів?	Як можна розвинути систему для доцільного портування та підтримки системи?	Як можна розвинути систему для найефективнішого виходу завдання?
Доступність	Як може бути розвинута система для вільного використання людьми з інвалідністю?	Як може бути розвинута система для вільного використання людьми з інвалідністю?	Як може бути розвинута система для вільного використання людьми з інвалідністю?	Як може бути розвинута система для вільного використання виходу завдання людьми з інвалідністю?

У кожного з цих типів користувач має потреби в якості у використанні і якості продукції в конкретних умовах використання, як показано на деяких прикладах користувачів і якісних характеристик з питань, показаних в (табл. 3.2).

До розробки програмного забезпечення або придбання, вимоги до якості повинні бути визначені з точки зору зацікавлених сторін. Аналіз вимог використання призведе до отриманих функціональних та якісних вимог, необхідних продукту для досягнення вимог використання.

Приклад потреби в надійності системи може привести до конкретних вимог для програмного забезпечення погашення якості, надійності, відмовостійкості і можливості відновлення. Надійність також може вплинути на загальну ефективність системи, ефективність, свободу від ризику та її насичення.

3.2.6.Відносини між моделями

Властивості програмного продукту та комп'ютерної системи визначення якості продукції в конкретному контексті використання.

Функціональна придатність, ефективність продуктивності, зручність використання, надійність і безпека матиме значний вплив на якість у використанні для основних користувачів. Ефективність, продуктивність, надійність і безпека можуть також бути конкретними проблемами інших зацікавлених сторін, які

спеціалізуються в цих областях.

Сумісність, ремонтопридатність і портативність матиме значний вплив на якість у використанні для вторинних користувачів, які підтримують систему.

Таблиця 3.3

Вплив якісних характеристик

Властивості програмного продукту	Властивості Комп'ютер оної системи	Якісні характеристики продукту	Вплив на якість при використанніі для превинних користувачів	Вплив на якість При використанні для обсулговуванн я завдач	Вплив на якість інформаці йної системи для іних зацікавлен их сторін
–	–	Функціональна придатність	+		
–	–	Рівень продуктивності	+		+
–	–	Сумісніть		+	
–	–	Зручність використання	+		
–	–	Надійність	+		+
–	–	Захищеність	+		+
–	–	Супроводжуваність		+	
–	–	Портативність		+	
«-» - властивості, які впливають на якість продукту					
«+» - якість продукції впливає на використання для інших зацікавлених сторін					

Сумісність, ремонтопридатність і портативність матиме значний вплив на якість у використанні для вторинних користувачів, які підтримують систему.

ВИСНОВОКИ ДО РОЗДІЛУ 3

Запропоновано нову модель якості оцінки якості ERP-систем у вищих навчальних закладах. Коефіцієнти якості та субфактори моделі були засновані на стандарті ISO 25010. Ця модель є першою у своєму роді, запропонованою для оцінки ERP-систем у ВНЗ. Дослідження показує три важливі внески, які включають порівняння двох популярних моделей якості програмного забезпечення, визначення

факторів якості ERP-систем та стандарту ISO 25010 як основи для нової моделі якісних ERP-систем у ВНЗ. Подальші дослідження щодо встановлення взаємозв'язків між факторами якості та субфакторами цієї моделі у ВНЗ будуть представлені в наступній статті. Результати подальшого дослідження дозволять нам зрозуміти взаємозв'язок і вплив цих субфакторів на фактори якості цієї моделі.

РОЗДІЛ 4
ВИКОРИСТАННЯ АРХІТЕКТУРНОГО ПРОЄКТУВАННЯ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ПП

4.1. Продукт архітектурного проектування

Архітектура ─ це давня метафора для проектування та розробки ПЗ, зокрема, для великомасштабного програмування, яка виникла у Фреда Брукса в 1960-х роках.

Архітектура ПЗ є важливою наукою в галузі інженерії ПЗ, але для різних людей архітектура може означати різні речі. Вона є формою будь-якої системи, створеної через свідоме проектування, і таким чином вона має сильні людські елементи як у процесі, так і в своєму продукті. Термін форма передбачає глибоку духовну модель сутності якоїсь структури. Структура має форму; дана форма чекає на реалізацію в структурі.

Архітектура називає те, що є фундаментальним або об'єднує систему в цілому; сукупність істотних властивостей системи, що визначають її форму, функцію, вартість, значення і ризик. Архітектура ─ фундаментальні поняття або властивості системи в її середовищі, що втілюються в її елементах, відносинах і в принципах її проектування та еволюції.

Архітектура ПЗ може характеризувати багато різних систем з потенційно різними функціями, реалізованими в різних мовах програмування. Форма - це глибока суть того, що є спільним між цими системами. Насправді, наявність структури заплутує форму з відволікаючими деталями і несуттєвими елементами. Архітектура спонукає до сутності системи.

Архітектура будь-якої системи є тим, що є суттєвим для даної системи в її оточенні. Не існує єдиної характеристики того, що є істотним або основним для системи, така характеристика може випливати з наступних аспектів:

- системні компоненти або елементи;
- взаємозв’язок та розміщення системних елементів;
- принципи організації системи або проекту;
- принципи, які керують розвитком системи в її життєвому циклі.

Важко стверджувати, що існує широко прийняте визначення архітектури у галузі ПЗ. Одне з перших визначень архітектури ПЗ було надано Перрі та Вульфом у 1992 році. Вони визначають архітектуру таким чином:

Відповідно до цього визначення, це є поєднання набору архітектурних елементів, форми цих елементів як принципів управління взаємовідносинами між елементами та їх властивостями, і обґрунтування вибору елементів та їх форми в певному випадку. Це визначення стало основою для початкового дослідження в області архітектури ПЗ.

Басс, Клементс та Кантзман визначили архітектуру ПЗ таким чином: архітектура ПЗ - це набір структур, необхідних для роз'яснення щодо системи, які складаються з елементів ПЗ, відносин між ними та властивостей обох.

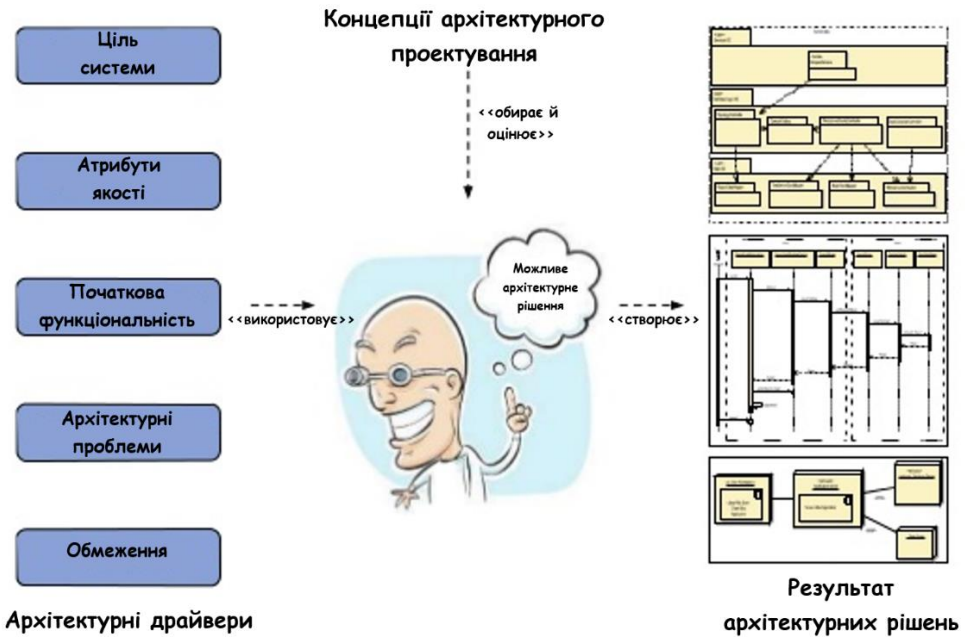

Рис. 4.1 Продукт архітектурного проектування.

Структури в архітектурі ПЗ представляють собою рішення про розподіл та взаємодію, що приймаються для розподілу обов'язків задовільняючих вимог між набором компонентів та визначення взаємозв'язків компонентів. Структурний поділ керується специфічними вимогами та обмеженнями застосування. Одне з головних міркувань під час вирішення розподілу полягає у створенні вільно пов'язаної архітектури з набору високозв’язних компонентів для мінімізації залежностей між ними. Структурне розбиття повинно здійснюватися як функціональними вимогами, так і нефункціональними. Кожна архітектурна структура може допомогти архітекторам зрозуміти різні атрибути якості системи. Архітектурні структури задокументовані з використанням різних архітектурних поглядів.

Програмна архітектура програми або обчислювальної системи є зображенням системи, яка допомагає зрозуміти, як система буде поводитися. Архітектура ПЗ служить планом як для системи, так і для проекту, що розробляє її, визначаючи робочі завдання, які повинні виконуватися командами проектування та впровадження. Архітектура є основним носієм системних якостей, таких як продуктивність, змінюваність і безпека, жодна з яких не може бути досягнута без уніфікованого архітектурного бачення. Архітектура є артефакт для раннього аналізу, щоб переконатися, що підхід до проектування дасть прийнятну систему. Створюючи ефективну архітектуру, ви можете визначити ризики дизайну та пом'якшити їх на початку процесу розробки.

Тобто, при створенні ПЗ архітектори займаються питаннями принципового розподілу великої програмної системи. В область їхніх інтересів входять ПЗ інфраструктури, яке надає механізми зберігання даних, розподіл розрахунків, обробки потоків, обміну повідомленнями, обробки транзакцій і забезпечення безпеки. Вони користуються політиками, шаблонами і потужними засобами, які дозволяють різним групам розробників створювати узгоджені системи, які є легкими для розуміння та супроводження. У цьому розумінні, архітектура ― це вираження погляду головного архітектора чи архітекторів.

4.2. Процес архітектурного проектування

Важливо також мати гарне розуміння процесу проектування архітектури та життєвого циклу архітектури. При проектуванні та оцінці архітектури ПЗ для масштабної, складної системи, важливо, щоб був дисциплінований процес, який може підтримувати творчість більш керованим та ефективним підходом. Крім того, як і будь-який інший артефакт, архітектура також має життєвий цикл, який проходить через різні етапи та заходи. Кожна фаза життєвого циклу архітектури має свої передумови для використання та застосування.

Нижче коротко описується кожна з видів діяльності в загальній моделі архітектури:

а) Аналіз проблемної області. Ця діяльність спрямована на визначення проблем, які потрібно вирішити. Основним видом діяльності може бути вивчення архітектурних вимог, з метою відокремлення та визначення пріоритетності архітектурно значимих вимог від тих, які не є архітектурно значимими.

б) Розробка та опис архітектурного рішення. Архітектор може розглянути кілька можливих варіантів проектування, перш ніж вибирати ті, які є найбільш доцільними та оптимальними. Архітектор також відповідає за документування розробленої архітектури, використовуючи відповідні позначення документації та шаблони.

в) Архітектурна оцінка. Ця діяльність спрямована на те, щоб архітектурні рішення, обрані під час попереднього процесу, були правильними.

г) Реалізація архітектури. Це фаза, в якій спроектована архітектура деконструюється у детальному проекті та реалізується. На цьому етапі розробники ПЗ складають кілька десятків рішень, які повинні бути узгоджені з рішеннями архітектури високого рівня.

д) Технічне обслуговування (еволюція) архітектури. Це фаза, яка передбачає внесення архітектурних змін, коли архітектура розвивається завдяки вимогам.

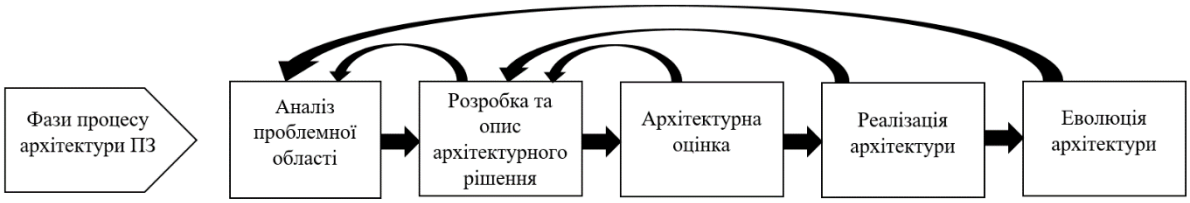

Рис. 4.2 Модель процесу архітектури програмного забезпечення.

Слід зазначити, що вище описані заходи не виконуються послідовно, як модель водоспаду. Навпаки, ці заходи виконуються досить ітеративно або еволюційно, і завдання, пов'язані з однією конкретною діяльністю, можуть бути виконані та/або переглянуті під час виконання будь-якої іншої діяльності.

Архітектуру називають "концепцією системи найвищого рівня у своєму середовищі" і необхідно розуміти, що архітектура охоплює такі аспекти, як цілісність системи, економічну доцільність її реалізації, естетику програмування і стиль. Тобто, в рамках архітектури розглядаються не тільки внутрішні елементи системи, але і взаємодія системи із зовнішнім середовищем, включаючи призначене для користувача середовище і середовище розробки.

4.3. Методи використання архітектурного проектування в гнучких технологіях

Архітектуру ПЗ розуміють як результат, що включає в себе набір комплексної системи рішень архітектурного проектування, що приймаються під час ітеративного та інкрементного процесу та залежать один від одного.

Як зазначалося раніше, процес проектування архітектури поділяють на три етапи: архітектурний аналіз, синтез та оцінка. Вирівнювання цих етапів з метою використання їх з гнучкими технологіями розробки ПЗ викликало багато проблем, пов’язаних із тим, що гнучкі підходи мають забезпечувати можливість прийняття змін, а архітектура сприймалася як план побудови системи. Не дивлячись на це, організації, що займаються розробкою ПЗ у гнучкому контексті, мають архітекторів.

З цього приводу існують дослідження, які вивчають застосування архітектурних технік в командах, що дотримуються гнучких технологій. Результати показали, що опитувані команди використовували в основному чотири принципи архітектурного проектування: big-up-front-design, sprintzero, in-sprints, separate-architecture-team.

В практиці **big-up-front-design** аналіз, синтез і оцінювання виконуються архітекторами до виконання спринтів, а не командою розробників. Проте на етапі впровадження системи в спринті все ж таки можуть вноситися певні невеликі зміни в проект архітектури.

У практиці **sprintzero** аналіз і синтез архітектури виконується у першому спринті командою розробників, а оцінка може проводить як у межах спринту так і після нього. Відмінністю від попередньої практики є те, що етап проектування є коротшим, а сам дизайн виконують розробники.

В практиці **in-sprints** архітектура будується в рамках спринтів командою розробників, при цьому архітектура реорганізовується в межах спринтів щоразу, коли виникає необхідність в цьому. Архітектурний аналіз здійснюються власником продукту або командою розробників, але в першому випадку процедура виконується за межами спринтів, у другому ― у самому спринті. Оцінювання тут як правило не проводиться, тому що внесення необхідних змін в готовий продукт вимагає великих затрат. Застосування цієї практики можливе лише висококваліфікованою і досвідченою командою.

В підході **separate-architecture-team** використовується окрема команда, завданням якої є розробка архітектури. В команду можуть входити представники різних команд розробників, в тому числі і архітектор. Збори цієї команди проводяться, коли це є необхідним, з метою проведення аналізу та синтезу архітектури, а фактична реалізація здійснюється командою розробників. Архітектурна оцінка проводиться командою при випуску нової версії.

Як слідує з наведеного аналізу, елементи архітектурного проектування ПЗ певним чином використовуються в гнучких технологіях. Але з метою досягнення більшого ефекту від застосування проектування архітектури для отримання високої

якості продукту, треба підвищити роль архітектора в гнучких практиках. Не слід нехтувати й привнесенням змін в архітектурні методи, наприклад, проектування архітектури, адаптованої до змін з метою полегшення внесення в неї корегувань при зміні вимог.

Зазначеного вище удосконалення практик можна досягти, проектуючи архітектуру у два етапи. На першому етапі виконується архітектурне проектування і вибір базової архітектури, а на другому - реалізується процедура «гнучкої» зміни архітектури. Для забезпечення мінливості архітектури пропонується на етапі аналізу виявляти «точки змін» та конструювати «профілі змін» на основі відповідей на запитання контрольного списку від усіх причетних до розробки ПС. «Контрольний список» виглядає наступним чином.

Таблиця 4.1

Питання контрольного списку для зацікавлених осіб.

Зацікавлені особи	Питання «контрольного списку
Усі учасники	- Чи відображає обрана архітектура інтереси зацікавлених сторін у створеній архітектурі? - Чи представляє обрана архітектура інтереси, що не є інтересами суб'єктів області? - Чи архітектура відповідає правилам та стандартам в області? - Чи можливо, щоб архітектура була інстальована в конкретних архітектурах продукту протягом часу та бюджету?
Архітектор	- Чи був чітко визначений час прив'язки кожної «точки змін»? - Чи чітко визначені та простежуються залежності від змін та обмеження на продукт? - Чи спільні та специфічні вимоги розділені і легко ідентифікуються? - Чи є опис потенційних контекстів, в яких вказуються архітектури? - Чи існують рекомендації для архітекторів та розробників щодо вирішення змін?

Продовження таблиці 4.1

Експерт галузі	- Чи чітко визначені пріоритети галузевих завдань, які система повинна задовольняти? - Чи ясно як визначаються вимоги галузевих цілей? - Чи є можливість відслідковування між цілями галузі та технічними рішеннями (тобто, чи можна переходити від цілей галузі до архітектурно значимих вимог, до технічних рішень)?
Менеджер програмного забезпечення	- Чи достатньо архітектурного опису, щоб оцінити зусилля для його реалізації? - Чи можна визначити залежності розвитку між різними частинами архітектури? - Які ресурси потрібні для демонстрації архітектури? - Чи існує графік впровадження та інтеграції?
Дизайнери та інтегратори	- Ви розумієте «точки змін» та варіанти в архітектурі? - Чи можете ви визначити підходи для здійснення змінності? - Чи можете ви визначити критерії успіху для тестування? - Чи можна визначити "представницькі" проблеми, що виникають при тестуванні архітектури?

Але у цьому випадку внесення змін в архітектуру і оцінювання відповідності її якості вимогам відбуватимуться шляхом аналізу відповідей експертів на запитання «контрольного списку», а це є досить суб’єктивним і не точним підходом.

Для оцінювання якості архітектури пропонується також використовувати методи сценаріїв як на першому, так і на другому етапах. Однак, ці методи є дуже трудомісткими, і їх застосування може потребувати до одного місяця, що є неприпустимим, особливо на другому етапі.

Також для поєднання архітектурного проектування і гнучких методів розробки є можливим контролювати якість зміненої архітектури шляхом обчислення цільової функції. Це дозволить досить швидко отримати оцінки якості і прийняти рішення про продовження процедури гнучких технологій, або перехід на процедуру перепроектування архітектури. На першому етапі формуються вимоги до ПС, далі

визначені вимоги формуються як вимоги до архітектури, створюються альтернативні архітектури на основі визначених критеріїв якості, які і є оцінками дотримання певних вимог, виконується оцінка (наприклад, методом аналізу ієрархій) та вибір архітектури.

На другому етапі в ітераціях гнучких технологій проводиться впровадження архітектури. В разі виявлення нових чи модифікованих вимог до ПС вносяться певні зміни в архітектуру, шляхом модифікації коду патернів архітектури або їхньої заміни на альтернативні. Наступний захід виконується з метою корекції значень якісних характеристик для оптимізації внесених змін. При внесенні суттєвих змін в архітектуру виконується оцінювання і перевірка відповідності якості зміненої архітектури вимогам. При незадовільній якості відбувається перехід на операції першого етапу, перепроектування і заміна базової архітектури.

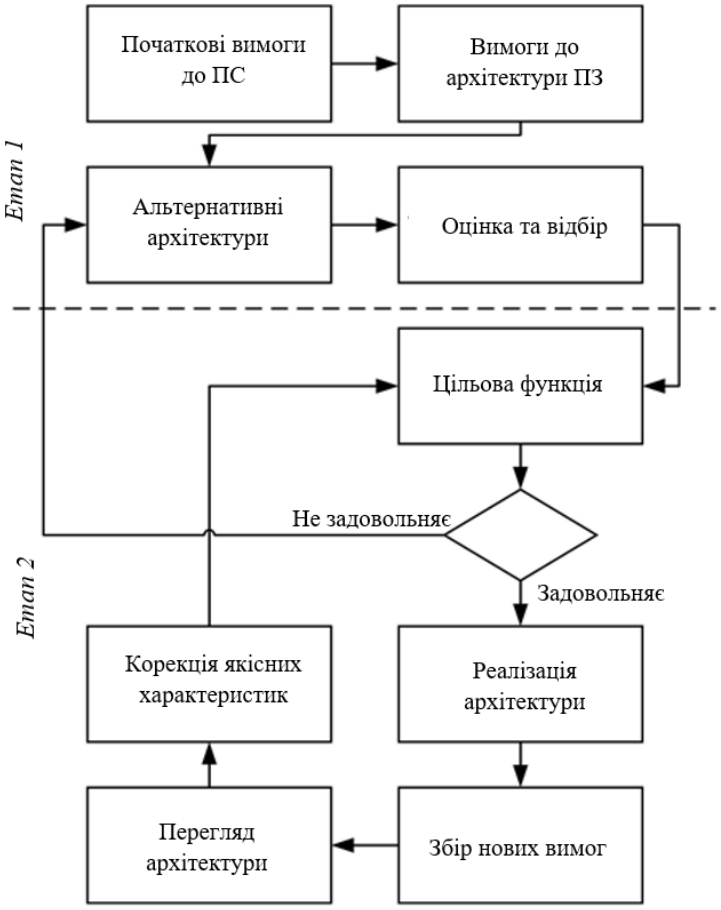

Рис. 4.3 Схема процесу поєднання проектування архітектури ПЗ з гнучкими підходами.

Отже, поєднання класичного архітектурного проектування з гнучкими підходами є одним із шляхів до вирішення проблеми забезпечення якості, що є актуальною проблемою у зв'язку з широким застосуванням гнучких методів проектування ПЗ.

4.4. Гнучкі підходи розробки програмних продуктів у межах великомасштабних проектів

В останні роки спостерігаються значні зміни в тому, як розвивається програмне забезпечення з появою гнучких методів розробки. Вважається, що ці методи найкраще підходять невеликим групам розробників, які займаються не життєво критичним ПЗ. Проте, з популярністю гнучких методів, багато хто почав використовувати гнучкі технології і у великих проектах.

Отже, що ж таке «великомасштабна гнучка розробка»? Загальним визначенням великомасштабної гнучкої розробки є зусилля розробників з більш ніж двома командами розробки, які мають складні кордони знань в рамках програми як результат бізнес-завдань галузі, кількості цих завдань і залежностей між ними. Крім того, такі програми характеризуються складною взаємодією з безліччю залучених технологій і, як правило, великою кількістю зацікавлених сторін.

Чому «великомасштабна гнучка розробка» актуальне на сьогоднішній день питання? По-перше, глобальне зосередження на цифрових технологіях призвело до більшого усвідомлення важливості ПЗ, оскільки воно пронизує кожен сектор суспільства. По-друге, ранні дослідження гнучкого розвитку у великих масштабах вказують на такі важливі аспекти, як координація команд та роботи. По-третє, керівництво компаній стало більш усвідомлювати важливість ПЗ, що призводить до оновлення методів розробки з метою забезпечення конкурентоспроможності.

Отже, успіх гнучких методів для невеликих спільно розташованих команд надихнув організації на більш широке використання цих технологій для створення складних програмних систем, але необхідно розуміти, що великі проекти

представляють суттєвий ризик і часто пов'язані з перевищенням витрат, пізніми завершеннями та відвертими невдачами проекту.

4.5. Архітектурний інформаційних потік у великомасштабній розробці

Протягом останніх двох десятиліть гнучкі методи трансформувалися і принесли унікальні зміни в практику розробки ПЗ, наголошуючи на тісному співробітництві в команді, залученні клієнтів та толерантності до змін. Успіх гнучких методів для невеликих, спільно розташованих команд надихнув організації на більш широке використання їх для створення складних ПС. Масштабування гнучких методів створює нові виклики, такі як координація між командами, залежності від інших існуючих середовищ або розробка архітектури ПС.

Проектування архітектури, яке охоплює в собі архітектурні вказівки, необхідне для того, аби синхронізувати міжгрупове проектування та реалізацію, аби уникнути редизайну, архітектурної дивергенції та функціональної надмірності, що збільшує складність системи. Ефективна еволюція архітектури ПС вимагає правильного балансу архітектури та тісної співпраці й спілкування між архітекторами та командами розробників при використанні гнучких підходів до розробки ПЗ.

Під «спілкуванням» розуміється створення або обмін думками, ідеями і емоціями. Комунікацію можна розкласти на два типи: міжкомандна і внутрішньокомандна. Перший тип комунікації - це зв'язок між кількома командами, другий - для спілкування в команді. Потік зв'язку, що з'єднує співрозмовників, називається мережами зв'язку. На зображено п'ять загальних мереж зв'язку, які поширені на сьогоднішній день.

Колісна мережа (Wheel) є найбільш централізованим шаблоном мережі. У цій мережі кожен член спілкується лише з однією особою. Керівник С отримує всю інформацію від своїх підлеглих A, B, D і E й посилає назад інформацію, як правило, у формі рішень.

Мережа ланцюгів (Chain) є другою за величиною в централізації. Лише двоє людей спілкуються один з одним, і з ними може спілкуватися лише одна людина.

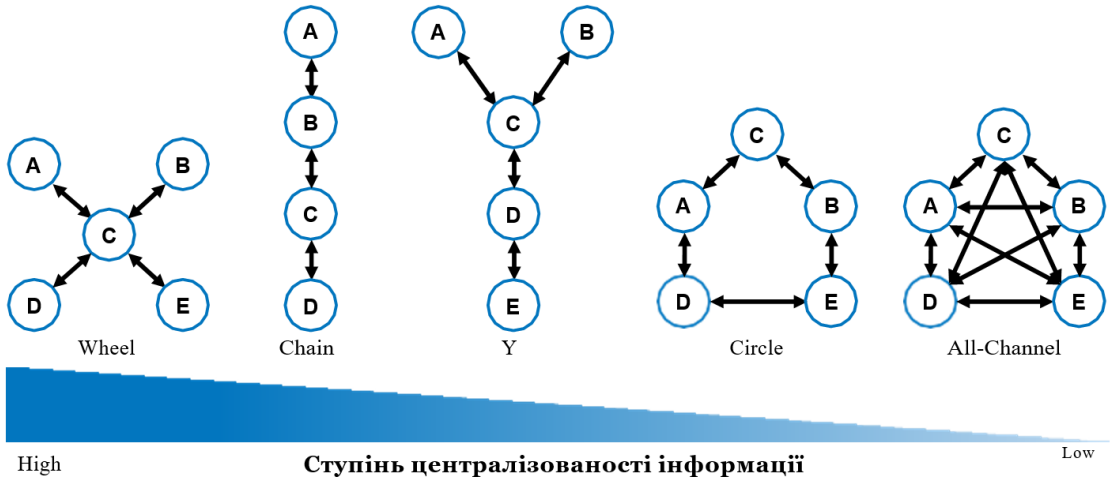

Рис. 4.4 Загальні комунікаційні мережі.

Мережа Y схожа на ланцюгову мережу, за винятком того, що два члени є поза ланцюгом. У мережі Y учасники A і B можуть надсилати інформацію в C, але вони не можуть отримувати інформацію від інших. Члени C і D можуть обмінюватися інформацією. Член E може обмінюватися інформацією з членом D.

Мережа-коло (Circle) виступає за горизонтальну і децентралізовану комунікацію, яка пропонує рівні можливості спілкування для кожного члена. Кожен може спілкуватися один з одним справа і зліва. Учасники мають однакові обмеження, але коло є менш обмеженою умовою, ніж колесо, ланцюг або мережа Y.

Мережа «всеканальних зв'язків» (all-channel network) є розширенням мережі кола і з'єднує кожного в мережі кола, оскільки вона дозволяє кожному члену вільно спілкуватися з усіма іншими особами.

Архітектори в Scrum часто стикаються з такими проблемами, як недостатня співпраця, нерозуміння цінності архітектури та погана комунікація між командними архітекторами. Як зазначалося раніше, у гнучких технологіях задля забезпечення якості розробки ПЗ вводять додаткові ролі, серед яких функціональний та технічний архітектор.

Кожен технічний архітектор, призначений для команди, береться за загальну архітектуру системи з її підсистемами та інтерфейсами, а команда, у свою чергу, концентрується на міжсистемних потоках даних і процесах, що пов'язані з інтеграцією архітектури. Ці потоки даних і процеси використовуються при

визначенні мінімальних вимог до інтерфейсу, яким повинні відповідати всі команди. Кожен підрозділ має також функціонального архітектора, який займається функціональною архітектурою підсистеми.

Обидві ролі архітекторів повинні відігравати подвійну роль у своїх підрозділах, роблячи архітектурні рішення і керуючи їх виконанням, відповідно до необхідних архітектурних стандартів. Причому, чим коротші часові інтервали між спостереженнями, тим більш домінуючим стає архітектор щодо обміну інформацією, причому як на рівні внутрішньо- командного обміну, так і на рівні між -командного обміну архітектурою.

Архітектори повинні підтримувати також і децентралізований обмін інформацією, що не відповідає схемам, зазначеним на, без обов’язкового залучення архітектора. І це відповідає цінностям та принципам технологій гнучкої розробки ПЗ. Як технічний, так і функціональний архітектори мають віддавати перевагу особистому спілкуванню з членами своєї команди.

На тему архітектруного інформаційного потоку є низка досліджень, в ході яких проводилось інтерв’ю у компаніях, що використовують Scrum у своїх проектах. Кожна з команд, що приймала участь в опитуванні, використовувала один з підходів до архітектурної роботи, що описувалися раніше: big-up-front-design, sprint-zero та in-sprints. Ці команди не використовували separate-architecture-team.

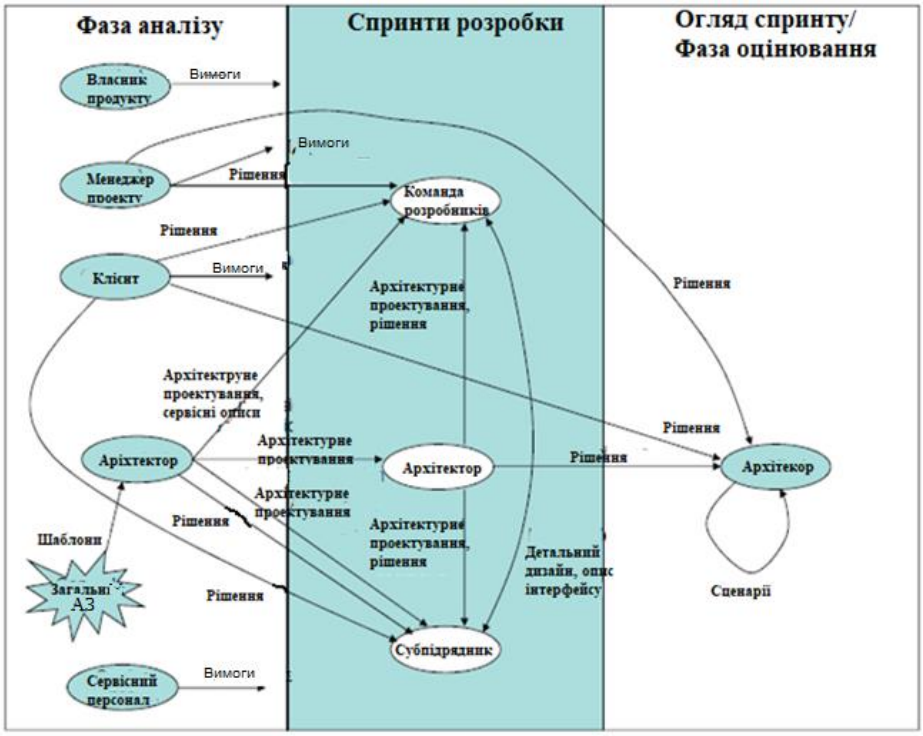

Рис. 4.5 Архітектурний інформаційний потік у підходах big-up-front-desigh та sprint-zero.

На (рис. 4.5) підсумовується потік архітектурної інформації в командах, що використовували підходи big-up-front-design та sprint-zero. Приймається, що перший спринт у підході sprint-zero є частиною фази аналізу.

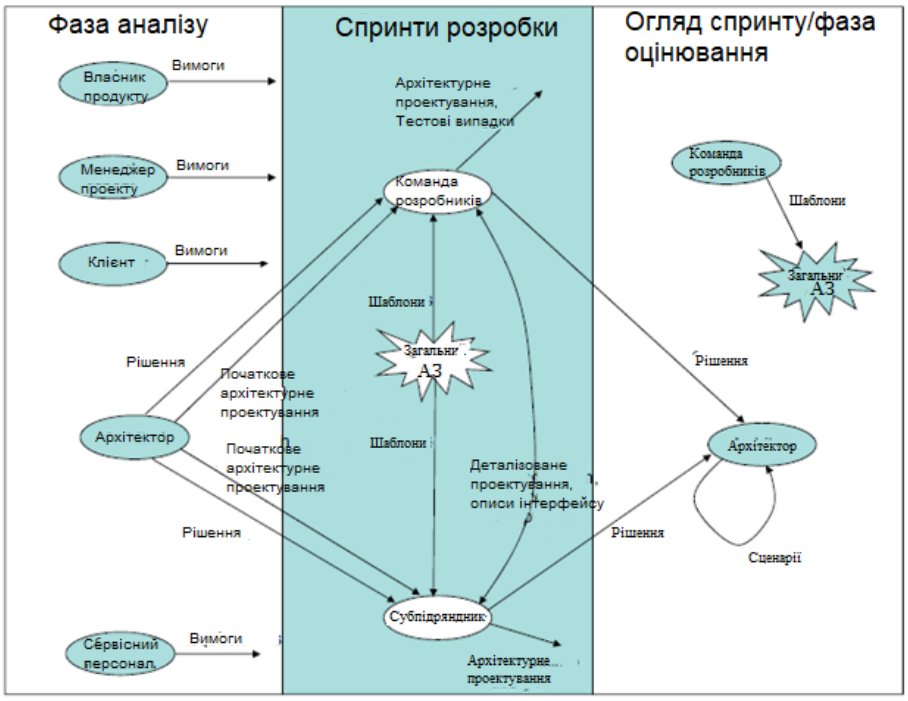

Рис. 4.6 Архітектурний інформаційний потік у підході in-sprints.

На (рис. 4.6) показує потоки архітектурної інформації в підході insprints.

Стрілки на рисунках зображають інформацію, яка видається зацікавленою стороною (виходить зі стріли) і споживається (входить стрілку) на певній фазі. Двох -направлені стрілки означають, що архітектурна інформація виробляється і споживається зацікавленими сторонами.

Відмінності в інформаційному потоці між big-up-front-design/sprint-zero та in-sprints чітко відображені: у першому - архітектурні рішення виробляються на етапі аналізу кількома зацікавленими сторонами і в основному споживаються в спринтах, тоді як в in-sprints - тільки початкове проектування архітектури виготовляється архітектором на етапі аналізу, і більшість архітектурних рішень та конструкцій виробляються під час спринтів.

Заслуговує уваги той факт, що вимоги до архітектури виробляються на етапі аналізу, але не споживаються явно на етапі розробки, як можна було очікувати (стрілки нікуди не йдуть). Пояснити це можна тим, що вимоги відображаються в архітектурних рішеннях і проектах та використовуються у цій формі під час розробки. Ще однією причиною може бути те, що при використанні Scrum вимоги відображаються у частині відставання продуктів (в історії користувачів). Тому ніхто не використовує вимоги явно. Шаблони використовувались архітектором на етапі аналізу (big-up-front-design) та командою розробників на етапі розробки (in-sprints).

Під час огляду можуть обговорюватися архітектурні шаблони з метою обговорення проблем, які виникали при роботі з ними, та виявлення дійсно працездатних із них. Отже, команда вдосконалює загальні архітектурні знання (АЗ) на цьому етапі. Команди також використовують у роботі й власні внутрішні моделі.

При оцінці архітектури використовуються рішення та сценарії, які також використовуються на етапі перегляду спринтів. Рішення, як правило, обговорюються при огляді спринтів під час навчальної сесії.

Згідно інформації, представленої на (рис. 4.5, 4.6), самі архітектурні проекти не використовуються при огляді спринтів, але обговорення зосереджується на архітектурних рішеннях. Сценарії розробляються та використовуються архітектором при оцінюванні архітектури.

Що стосується інтерфейсів, то навіть у підходах big-up-front-design/sprint-zero інтерфейси визначаються лише на етапі розробки. Це пов’язано із тим, що нераціонально вказувати інтерфейси наперед, замість цього слід просто описати, які послуги повинен забезпечити інтерфейс, оскільки проектування інтерфейсу завжди змінюватиметься під час виконання.

У in-sprints підході проектування архітектури виконується субпідрядниками та командами розробників, але не використовується жодною із зацікавлених сторін. Це пов'язано з тим, що продукти, з якими працювали опитані команди, ще не були випущені. Архітектурне проектування було вже розроблено та задокументовано для подальшого використання (наприклад, для етапу технічного обслуговування продукту). Однак зазвичай ця інформація, ймовірно, буде використана самими командами.

4.6. Проблеми забезпечення якості програмних продуктів в гнучких технологіях проектування

Проекти розробки ПЗ зазнали значних змін з приходом гнучких підходів до розробки. Виникнувши в 1990-х роках, вони перетворили і принесли безпрецедентні зміни до практики розробки ПЗ. Гнучкі підходи підкреслюють залучення клієнтів, технічну якість продукції, включаючи мінливі та нові вимоги, а також ідею про те, що розробка ПЗ найкраще виконується в невеликих командах, що керуються самостійно.

Парадигма гнучких технологій розробки ПЗ широко використовується сотнями великих та малих компаній, які прагнуть зменшити витрати та збільшити їх здатність керувати змінами в динамічних ринкових умовах, хоча завжди існував скептицизм щодо надійності, результативності та ефективності тих методів гнучких технологій, які не приділяють достатньої уваги важливим ролям, пов’язаних з принципами, практиками і артефактами архітектури ПЗ.

Зазвичай, найпоширенішими викликами, що постають перед компаніями при адаптації та масштабуванні гнучких технологій є (рис. 4.7):

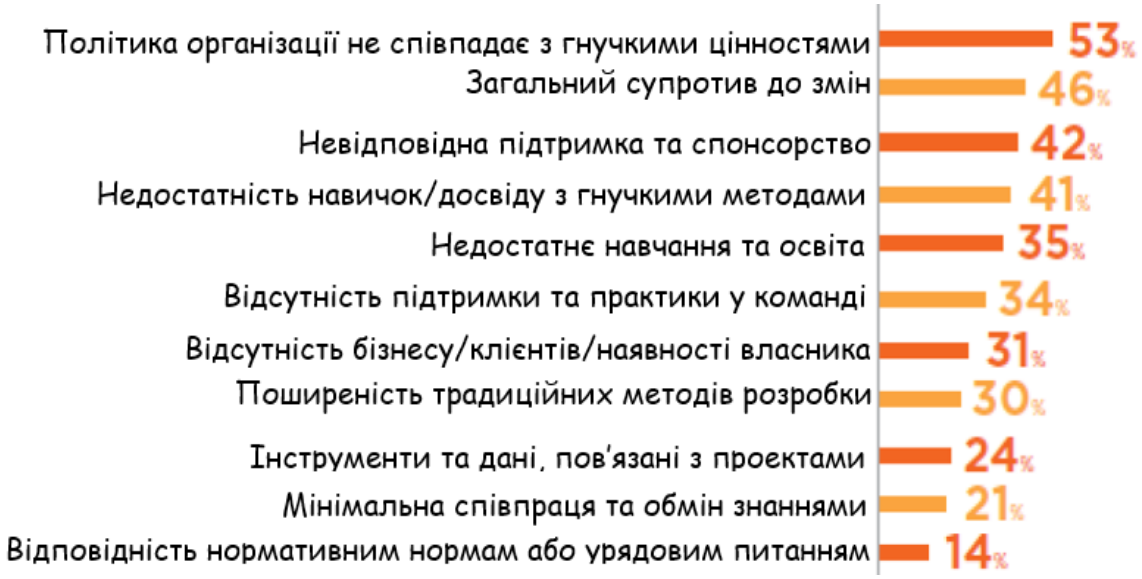

Рис. 4.7 Труднощі, які виникають під час адаптації гнучких технологій

4.7. Архітектура як інструмент забезпечення якості програмних продуктів

Широко визнається, що архітектура ПЗ може стати ефективним інструментом скорочення витрат, часу розвитку та еволюції й підвищити концептуальну цілісність та якість системи. Також робиться наголос на координації з метою планування, синхронізації та огляду й на практиці, щоб змусити команди ефективно працювати разом.

Отже, зростає розуміння того, що необхідно приділяти більше уваги архітектурним аспектам в гнучких підходах до розробки ПЗ, оскільки стали достатньо актуальними проблеми забезпечення їх якості. Таким чином, зростає кількість зусиль, спрямованих на виявлення технічних та організаційних проблем, пов'язаних з інтеграцією гнучких підходів у традиційні методи розробки ПЗ.

За словами прихильників гнучких технологій рефакторинг може допомогти усунути більшість системних проблем, пов'язаних із інтенсивними програмними системами. При цьому стверджується, що звичайний рефакторинг цілком доцільний, поки проект високого рівня є достатнім для обмеження потреби в масштабному рефакторингу. І багатий досвід показує, що масштабний рефакторинг часто призводить до значних дефектів, які приносять багато витрат для подальшої підтримки життєвого циклу розробки.

Основними перевагами гнучких методів є те, що вони пристосовані до врахування постійної зміни вимог, які аналізуються і уточнюються на початку кожної ітерації. За рахунок цього, а також проведенням неперервного тестування і інтеграції, зменшуються ризики проекту, збільшується його ефективність.

Однак, гнучкі методи мають і ряд недоліків, з яких слід виділити наступні:

- вимоги в гнучких методах є «полегшеними» і керованими локально в межах кожної ітерації, тому можуть з’явитися проблеми з їх стабілізації та узгодження при інтеграції;
- можливі також проблеми з необхідністю створення «гнучкої» архітектури, яку потрібно постійно корегувати, для врахування зміни вимог.

Більшість методів гнучких підходів приділяють дуже мало уваги загальноприйнятим заходам архітектурного проектування на початкових етапах розробки ПЗ, отримуючи архітектуру з коду, а не з передової структури. С. Хофмейстер виділяє у процесі архітектурного проектування три таких етапи: архітектурний аналіз, архітектурний синтез, архітектурна оцінка та типи артефактів, пов'язані з цими діями. На етапі аналізу, на основі вимог до програмної системи (ПС), формуються вимоги до архітектури. На етапі синтезу, на основі аналізу вимог до архітектури, будуються варіанти архітектури. На останньому етапі проводиться оцінювання варіантів архітектури по множині критеріїв якості та вибір кращої з них, яка реалізується на наступних етапах проектування ПС.

Дослідники починають визнавати, що обидві дисципліни (методи гнучких технологій та архітектурно-орієнтовані підходи) мають важливі та взаємодоповнюючі ролі у розробці ПЗ та еволюційній діяльності і слід більше приділяти уваги архітектурним аспектам у гнучких підходах.

Ця ситуація стимулювала зусилля, спрямовані на виявлення механізмів та передумов інтеграції відповідних архітектурно-орієнтованих принципів та практик в гнучких підходах.

У (табл. 4.2) відображає, що деякі відомі гнучкі практики разом з архітектурною практикою мають еквівалентні принципи і можуть бути легко пристосовані один до одного.

Таблиця 4.2

Поєднання гнучких та архітектурно-орієнтованих підходів

Деякі гнучкі практики	Архітектурно-орієнтовані практики
Спринт	Ітераційна природа загальної моделі архітектури ПЗ
Планування спринту	Пріоритетність архітектурно значимих вимог для кожної ітерації
Огляд спринту	Архітектурний огляд
Щоденні зустрічі	Спільне розуміння архітектури та знань життєвого циклу архітектури
«Клієнт на місці»	Залучення зацікавлених сторін у більшість фаз життєвого циклу архітектури
Безперервна інтеграція	Інтеграція та сумісність на рівні архітектури
Рефакторинг	Рефакторинг на рівні архітектури з використанням шаблонів і архітектурних стилів
Метафора	Розробка архітектури ПЗ
Простий дизайн	Дизайн на основі шаблонів, щоб зберегти дизайн простим і зрозумілим
Колективна власність коду	Участь зацікавлених сторін у вирішенні ключових архітектурних проектів
Стандартний код	Архітектурні шаблони та стандарти для підтримки спільних цілей та стандартів
Тест-керована розробка	Тестування на основі архітектури

При інтеграції гнучких та архітектурно-орієнтованих практик архітектор має дотримуватись деяких ключових завдань, аби досягти успіху:

– Архітектор повинен добре розуміти гнучкі підходи.

– Архітектор повинен знати, як продавати ключове дизайнерське рішення власникам продукції в конфліктних ситуаціях.

– Архітектор проекту повинен знати загальну архітектуру, необхідні функції та статус реалізації.

– Архітектор повинен документувати та передавати архітектуру всім зацікавленим сторонам.

– Архітектор повинен бути готовий займати багато архітектурних посад - архітектор рішення, архітектор ПЗ та архітектор впровадження.

– Архітектор повинен очолити зусилля для інституціоналізації ролі архітекторів як фасилітаторів та постачальників послуг у проектах.

Беручи до уваги програму Perform норвезького пенсійного фонду, яка є прикладом великомасштабного проекту і була розроблена з метою оновлення системи автоматизації офісу, і оцінюючи їх досвід, можна виявити використання додаткових ролей, таких як - технічний архітектор, архітектор рішень, функціональний архітектор і відповідальний за тестування, а також кілька додаткових арен на додаток до Scrum of Scrum, що допомогло врегулювати численні залежності між роботою команд і забезпечити координацію.

Загалом, одна з найбільших програм розвитку в Норвегії, Perform, надає приклад того, як дванадцять команд Scrum поєднували гнучкі практики з традиційним управлінням проектами, висуваючи 12 ключових уроків (табл. 4.3).

Таблиця 4.3

Ключові концепції адаптації гнучких технологій до традиційного бачення управління проектами.

Назва концепції	Опис концепції
Процес відставання продукту	Необхідно спільно розробити загальний аналіз опису потреб та рішень, використовуючи при цьому метод набігаючої хвилі, з метою вибору історій користувачів, які є «достатніми», щоб отримати правильний рівень деталізації в цьому процесі.
Загальне відставання	Всю область програми слід організувати в одному спільному відставанні продукту, оскільки окремі відставання продукту викликають складнощі при переносі історій користувачів з одного відставання в інше, а також ускладнює процедуру визначення пріоритетів.

Продовження таблиці 4.3

Безперервний опис рішень	Історії користувача описуються під час роботи над одним релізом. Це забезпечує ефективне використання ресурсів у описі рішень, оскільки описуються лише історії користувачів, які збираються реалізовувати.
Різний рівень деталей	Команди, що описують тонкощі в різних деталях, випереджають, концентруючись на більш відкриті уточнення, але постійну співпрацю для вирішення деталей
Додаткові ролі	Додаткові ролі встановлюються на початку програми та впроваджуються у всіх командах розробників. Очікується, що кожна команда матиме технічного архітектора, відповідального за технічне проектування, функціонального архітектора, відповідального за описи рішень, тест-відповідального і розробників різних рівнів підготовки. Ця матрична організація, де члени команди працюють і в крос-командних проектах, має кілька переваг, включаючи економію часу, усне спілкування та досягнення відчуття «спільної роботи над цим».
Додаткові арени	Для того, щоб підтримувати швидкість передачі історій користувачів, важливо збільшити координацію та комунікацію команд та постачальників. Можлива організація щоденних зустрічей в групах розробників, засідань в рамках кожної команди, засідань для керівників проектів і менеджерів підпроектів, а також проведення ретроспектив. Крім того, ефективними є формальні та неформальні арени для координації, навчання та стандартизації.
Тестовий проект	Тестування можливо організувати як окремий проект з ресурсами від усіх команд розробників на додаток до менеджера командного зовнішнього проекту, тестувальників і тестових менеджерів для кожного з підпроектів розвитку.

Продовження таблиці 4.3

Процес затвердження релізів	Новий реліз можливо пропускати через процес затвердження в програмі перед передачею на приймальне тестування. Цей додатковий процес необхідний тому, що під час останньої ітерації часто важко перевіряти більш довгі вартісні ланцюжки, а процес затвердження надає більше уваги не функціональним вимогам, таким як працездатність, надійність і продуктивність.
Ретроспективи	Всі команди зобов'язані проводити ретроспективи в кінці кожної ітерації і протоколювати це. Усі протоколи мають бути прочитані центральним керівництвом і цей відгук від команд є основою для впровадження змін, а також для щотижневих оцінок ризику.
Демо як навчальна арена	Командам було надається 10 хвилин, щоб продемонструвати прогрес після кожної ітерації і для перегляду запрошуються усі зацікавлені сторони.

4.8. Необхідність координації в процесах узгодженості

Використовуючись до цього для невеликих команд, гнучкі підходи все частіше використовуються в інших умовах, наприклад, у великих програмах з декількома групами. Великі програми зазвичай включають технічну та організаційну складність. Сюди входить велика кількість зацікавлених сторін, велика кількість учасників програми, велика кількість вимог, рядків програмного коду, а часто навіть дуже складні взаємозалежності між завданнями, так само як між командами, які залежать від інших команд. Програми, що використовують гнучкі підходи, ризикують відсутністю взаємодії та труднощами у спілкуванні, тому що більшість комунікацій здійснюється усно в команді. Така складність загалом негативно впливає на роботу проекту. Масштабні програми становлять більший ризик і часто пов'язані з перевитратами, пізніми завершеннями та невдачами проекту.

Серед чинників, які наголошують на необхідності координації з метою налагодження взаємодії виділяють:

– Невизначеність завдання - складність і мінливість роботи, що виконується організаційною одиницею. Більш високі ступені складності, час мислення для вирішення проблем, або час, необхідний перед відомим результатом, свідчать про більш високу невизначеність завдання.

– Завдання взаємозалежності - ступінь, в якій люди в організаційній одиниці залежать від інших для виконання своєї роботи. Високий рівень співпраці, пов'язаного з виконанням завдань, означає високу взаємозалежність.

– Розмір робочої одиниці - кількість людей у робочій одиниці. Збільшення учасників проекту або програми означає збільшення розміру робочого підрозділу.

На сьогоднішній день методи, розроблені раніше для окремих команд з 5-9 розробників, були адаптовані для використання в проектах з десятками команд, сотнями розробників, які можуть включати інтеграцію з сотнями існуючих систем і впливати на сотні тисяч користувачів. Таким чином, почали з’являтися методології для керування великими гнучкими проектами розробки, такі як Scaled Agile Framework, Large-Scale Scrum і т.д. Деякі з нових методологій дають широкі рекомендації щодо ряду областей, такі як Scale Agile Framework, в той час як інші, такі як Spotify, просто рекомендують більше повноважень щодо прийняття рішень надати автономних командам.

ВИСНОВКИ ДО РОЗДІЛУ 4

Отже, існують деякі складнощі в організації спільної роботи гнучких та архітектурних підходів з метою розробки систем безпеки, бізнесу, захисту або критично важливих програмних систем та послуг. Якість, продуктивність та рентабельність можуть бути збільшені шляхом підвищення ефективності та дієвості процесів розробки ПЗ компаній. Загалом, масштабування гнучких методів створює нові виклики, такі як - координація між командами, залежності від інших існуючих середовищ або розподіл роботи без архітектури. І останнє є причиною критики великомасштабного гнучкого розвитку, оскільки нехтує допомогою архітектури у розробці ПЗ, що є ефективним лише на рівні команді, але недостатнім для великих

масштабів. Це, у свою чергу, може призвести до надмірних зусиль з редизайну, архітектурної дивергенції та функціональної надмірності, що збільшує складність системи.Перша хвиля гнучких методів була орієнтована безпосередньо на розвиток команд, акцентуючи увагу на розробці високопріоритетних функцій, призначення декількох ролей і легких у використанні артефактів для сприяння розробці. Наразі, друга хвиля гнучких методів спрямована на вирішення завдань масштабу, ставлячи перед собою питання, пов’язані з багаторівневими організаціями, збільшенням ризиків, кількостей ролей і практик для координації та узгодження між командами.

Великомасштабна гнучка розробка залучає численну кількість людей у команди розробки, а це тягне за собою проблеми, пов’язані з ускладненням забезпечення узгодженості між ними та із забезпеченням співробітництва з клієнтами, оскільки великомасштабна розробка часто включає в себе велику кількість зацікавлених сторін. Отже, центральним питанням є застосування гнучких методів до великомасштабної розробки. Друге питання полягає у тому, як адаптувати гнучкі методи для масштабної розробки.

ВИСНОВКИ

Отже, можна сказати, що на на сьогоднішній день існують деякі труднощі щодо вирівнювання архітектури та гнучкої розробки. У Scrum, наприклад, немає явної ролі архітектора, але багато організацій розроблюють наперед проект і мають архітекторів, що в подальшому потребує передачі проекту команді розробників, а це важко здійснити. Архітектурне проектування не може бути повною мірою передано з використанням традиційних архітектурних документів, оскільки у багатьох випадках йому бракує чогось ще, окрім того, що вже спроектовано. Наприклад, реалізація може відповідати проектуванню і функціонувати належним чином, коли система має порядок 100 параметрів. Але коли кількість використовуваних параметрів значно зростає протягом життєвого циклу продукту, така ж реалізація вже не є прийнятною. Тому надзвичайно важливо надати команді розробників обґрунтування проекту та чітко вказати на основні проблеми дизайну.

Загалом, якщо проектування виконується без одночасної реалізації, існує велика ймовірність того, що спроектовану частину необхідно буде змінювати. Нова інформація з'являється під час впровадження, що робить вже спроектоване недійсним. Також дуже часто буває так, що клієнт хоче змінити свої вимоги, і нинішня спроектована частина не може впоратися з цими змінами.

Існує безліч архітектурних знань, які слід повідомляти та обговорювати у команді. У багатьох випадках команди розробників мають досвід роботи з галуззю та подібними системами, тому всі знання не потрібно повідомляти від архітектора до команди. Проте, якщо залучено чимало субпідрядників, або групи розробників змінюються, як рукавички, то АЗ повинні бути повідомлені. У цьому процесі архітектурна документація може допомогти, але не повністю замінити зв'язок віч-на-віч. Крім того, якщо команда розробників реалізує лише один компонент великої системи, вони можуть втратити зв'язок з архітектурним контекстом і можуть не враховувати всю інформацію, яка стосується компоненту. Повідомлення та обговорення архітектурної інформації є найбільш домінуючим етапом на початку спринтів.

Отже, через складність розробки великомасштабних продуктів, кожен підрозділ керується і підтримується принаймні одним технічним та функціональним архітектором. Кожен технічний архітектор відповідає за архітектуру підсистеми і забезпечує відповідність певної команди визначеним архітектурним вимогам. Роль функціонального архітектора відповідає за розробку функціональної архітектури підсистеми. Можна зробити висновок, що обидва архітектори повинні відігравати подвійну роль у своїх підрозділах. З одного боку, вони приймають архітектурні рішення й ітеративно створюють моделі архітектури. З іншого боку, вони забезпечують керівництво та підтримують свою бригаду у виконанні архітектурних стандартів. Майже у всіх мережах обміну інформацією архітектори формують центральні вузли в рамках між- та внутрішньо-командного координування інформації, при цьому технічний і функціональний архітектори віддають перевагу прямому спілкуванню. Зокрема, архітектори щоденно обмінюються інформацією зі своїми командами. Внутрішньо командний обмін між архітекторами та їх командами зазвичай характеризується мережею загального каналу зв'язк.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abrahamsson P. Agility and architecture: can they coexist? / Ali Babar M., Kruchten P., IEEE Softw 2010. pp.16–22.
2. Nord R.L. Software architecture-centric methods and agile development / Tomayko J.E. IEEE Softw 2006. pp.47–53.
3. Agile alliance: manifesto for agile software development [Electronic resource]. – http://agilemanifesto.org / (lastaccess:10.01.2010). – Title from the screen.
4. Versionone. [Electronic resource]. – http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Result.pdf / (lastaccess:29.08.2012). – Title from the screen.
5. Clements P. Evaluating software architectures. / , Kazman R. Klein M. Boston, MA: Addison-Wesley; 2002. pp.33-69.
6. Polarion software. [Electronic resource]. – http://www.polarion.com / (lastaccess:05.12.2000). – Title from the screen.
7. Lohan, G., Conboy, K., and Lang, M. 2010. "Beyond Budgeting and Agile Software Development: A Conceptual Framework for the Performance Management of Agile Software Development Teams," Proceedings of the 31th International Conference on Information Systems (ICIS) St. Louis, MO.
8. Mahadevan, L., Kettinger, W. J., and Meservy, T. O. 2015. "Running on Hybrid: Control Changes When Introducing an Agile Methodology in a Traditional" Waterfall" System Development Environment," Communication of the Association for Information Systems (36:5), pp. 77-103.
9. McHugh, O., Conboy, K., and Lang, M. 2012. "Agile Practices: The Impact on Trust in Software Project Teams," IEEE Software (29:3), pp. 71-76.
10. Haoues, M., Sellami, A., Ben-Abdallah, H. and Cheikhi, L. (2017). “A guideline for software architecture selection based on ISO 25010 quality related factors”, Int J Syst Assur Eng Manag (November 2017) 8(Suppl. 2):S886–S909

11. Iqbal, H. and Babar, M. (2016). “An Approach for Analyzing ISO / IEC 25010 Product Quality Requirements based on Fuzzy Logic and Likert Scale for Decision Support Systems”, International Journal of Advanced Computer Science and Applications,Vol. 7, No. 12
12. ISO/IEC (2011). ISO/IEC 25010 – “Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE)” - System and Software Quality Models. Technical report.
13. A. Presbitero, B. Roxas, D. Chadee, Эффекты внутри - и динамика межкоманды на организационном изучении: роль возможности обмена знаниями, Знание Справляется - ment Исследование и Практика 15 (1) (2017) 146–154.
14. .D. S. Cruzes, T. Dyba, Рекомендуемые шаги для тематического синтеза в разработке программного обеспечения, в: Эмпирическая Разработка программного обеспечения и Измерение (ESEM),2011 International Symposium on, IEEE, 2011, pp. 275–284.
15. A. Presbitero, B. Roxas, D. Chadee, Эффекты внутри - и динамика межкоманды на организационном изучении: роль возможности обмена знаниями, Знание Справляется - ment Исследование и Практика 15 (1) (2017) 146–154.