

***БАКАЛАВРСКА
РОБОТА***

БР.ІІ – 50.00.00.000 ІІЗ

Група ІІ-22-2

Гаргат Катерина

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Гаргат Катерина Юріївна

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

БАКАЛАВРСЬКА РОБОТА

Методи моделювання програмних процесів у Agile-середовищі

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня К.Ю. Гаргат
(підпис, ініціали та прізвище здобувача)

Науковий керівник Пасека Надія Мирославівна к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою

ІПЗ

доцент.

В.В. Бандура

“ ” 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Гаргат Катерині Юріївні

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) "Методи моделювання програмних процесів у Agile-середовищі"

керівник проекту (роботи) Пасєка Надія Мирославівна к.т.н., доцент.

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення

2. Аналіз вимог і постановка задачі

3. Проектування системи

4. Реалізація проекту

5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1 Взаємозв'язки між принципами, практиками, припущеннями та обмеженнями (рис. 1.3, ст. 23)

2 - Структура моделі залучення клієнтів (рис.2.2, ст. 45)

3 Планування та контроль (рис. 2.4, ст.47)

4 Вартість внесення змін (людино-днів/змін) у проєкті (рис. 3.2, ст. 60)

5 Якість проектування в проєкті (рис. 3.3, ст. 62)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 2026 р.

Керівник

_____ (підпис)

Завдання прийняв до виконання

_____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент

_____ Гаргат К.Ю
(підпис)

Керівник роботи

_____ Пасєка Н.М.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 95 сторінок, 16 рисунків, 11 таблиць, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів моделювання програмних процесів у Agile-середовищі та аналіз підходів до підвищення ефективності управління розробкою програмного забезпечення.

Об'єкт дослідження: процеси розробки програмного забезпечення в Agile-середовищі.

Предмет дослідження: методи та інструменти моделювання програмних процесів, включаючи системну динаміку, управління ресурсами та підходи до оптимізації гнучких процесів розробки.

Результати дослідження: проведено аналіз Agile-методологій, досліджено підходи до моделювання програмних процесів, розглянуто методи системної динаміки та інструменти управління розробкою програмного забезпечення.

У першому розділі розглянуто теоретичні основи Agile-підходів, передумови їх виникнення, а також базові принципи моделювання програмних процесів і припущення, що лежать в основі гнучких методологій.

У другому розділі досліджено методи та інструменти моделювання процесів розробки програмного забезпечення, включаючи системну динаміку, використання хмарних платформ і аспекти управління ресурсами в Agile.

У третьому розділі представлено результати моделювання, проведено аналіз ефективності застосованих підходів.

Висновок: застосування методів моделювання у Agile-середовищі дозволяє підвищити прозорість процесів розробки, покращити прогнозування результатів і підвищити загальну ефективність команд розробки.

КЛЮЧОВІ СЛОВА: AGILE, SCRUM, KANBAN, МОДЕЛЮВАННЯ ПРОЦЕСІВ, СИСТЕМНА ДИНАМІКА, УПРАВЛІННЯ ПРОЄКТАМИ, РОЗРОБКА ПЗ, ОПТИМІЗАЦІЯ.

ANNOTATION

The bachelor's thesis contains of 95 pages, 16 figures, 11 tables, and a reference list with 40 sources.

The aim of the work is to study methods for modeling software development processes in an Agile environment and to analyze approaches to improving the efficiency of software development management within flexible methodologies.

Research object: software development processes in an Agile environment.

Research subject: methods and tools for modeling software processes, including system dynamics, resource management, and approaches to optimizing Agile development processes.

Research results: an analysis of Agile methodologies was conducted, approaches to modeling software processes were studied, system dynamics methods and software development management tools were examined, and the effectiveness of applying models in a flexible environment was evaluated.

The first section describes the theoretical foundations of Agile approaches, the prerequisites for their emergence, as well as the basic principles of software process modeling and assumptions underlying Agile methodologies.

The second section analyzes methods and tools for modeling software development processes, including system dynamics, the use of cloud platforms, and aspects of human resource management in Agile teams.

The third section presents the results of modeling, evaluates the effectiveness of the applied approaches, and discusses the limitations of Agile processes and ways to overcome them.

Conclusion: the use of modeling methods in an Agile environment improves process transparency, optimizes resource utilization, enhances result predictability, and increases the overall efficiency of development teams.

KEY WORDS: AGILE, SCRUM, KANBAN, PROCESS MODELING, SYSTEM DYNAMICS, PROJECT MANAGEMENT, SOFTWARE DEVELOPMENT, OPTIMIZATION.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ПРОГРАМНИХ ПРОЦЕСІВ В AGILE-СЕРЕДОВИЩІ	9
1.1 Передумови розвитку Agile-методологій та огляд пов'язаних досліджень.	9
1.2 Основні підходи до моделювання програмних процесів	14
1.3 Припущення та принципи, що лежать в основі гнучких процесів розробк.	23
1.4 Висновки по розділу	40
РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ МОДЕЛЮВАННЯ В AGILE-СЕРЕДОВИЩІ	41
2.1 Моделювання системної динаміки процесів розробки програмного забезпечення	41
2.2 Інструменти та платформи підтримки Agile-процесів і їх моделювання	46
2.3 Моделювання процесів командної взаємодії та управління ресурсами в Agile	52
2.4 Висновки по розділу	57
РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТА ОЦІНКА ЕФЕКТИВНОСТІ AGILE-ПРОЦЕСІВ	58
3.1 Результати моделювання та експериментальних досліджень	58
3.2 Аналіз обмежень Agile-процесів та шляхи їх подолання	63
3.3 Оцінка ефективності моделей та інтерпретація отриманих результатів	74
3.4 Висновки по розділу	93
ВИСНОВКИ	94
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА	96

					БР.ІІ – 50.00.00.000 ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Методи моделювання програмних процесів у Agile-середовищі				Літ.	Арк.	Акрушіє
Розроб.		Гаргат К.Ю.									
Перевір.		Пасека Н.М.								7	
Реценз.		Вовк Р.Б.							ІФНТУНГ ІІ-22-2		
Н. Контр.		Піх М.М.									
Затверд.		Бандура В. В.			Пояснювальна записка						

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій розробка програмного забезпечення стає дедалі складнішим і динамічнішим процесом, що потребує швидкої адаптації до змін вимог, ринку та технологічного середовища. Традиційні підходи до управління розробкою поступово поступаються місцем гнучким методологіям, таким як Agile, які забезпечують ітеративність, гнучкість та орієнтацію на результат. Водночас із впровадженням Agile-підходів виникає необхідність у ефективних методах моделювання програмних процесів, що дозволяють прогнозувати результати, оптимізувати використання ресурсів і підвищувати якість кінцевого продукту.

Актуальність теми зумовлена зростаючою потребою у формалізації та аналізі процесів розробки програмного забезпечення в умовах гнучких методологій. Незважаючи на переваги Agile, такі як адаптивність і швидка реакція на зміни, ці підходи часто супроводжуються труднощами в плануванні, оцінці термінів виконання та управлінні ресурсами. Існуючі інструменти та методи не завжди забезпечують достатній рівень прозорості процесів, що ускладнює прийняття обґрунтованих управлінських рішень. Тому дослідження методів моделювання програмних процесів у Agile-середовищі є важливим кроком для підвищення ефективності розробки програмного забезпечення.

Метою роботи є дослідження та аналіз методів моделювання програмних процесів у Agile-середовищі з метою підвищення ефективності управління розробкою програмного забезпечення та оптимізації використання ресурсів.

Завданнями дослідження є аналіз теоретичних основ Agile-методологій і принципів моделювання програмних процесів, визначення ключових припущень, що лежать в основі гнучких підходів, дослідження сучасних методів і інструментів моделювання, зокрема системної динаміки, аналіз ролі хмарних платформ і управління людськими ресурсами, проведення моделювання програмних процесів і оцінка ефективності отриманих результатів, а також визначення обмежень Agile-процесів і шляхів їх подолання.

Об’єктом дослідження є процеси розробки програмного забезпечення в Agile-середовищі, що включають планування, реалізацію, тестування та управління проєктами.

Предметом дослідження є методи, моделі та інструменти моделювання програмних процесів, які застосовуються для аналізу, оптимізації та прогнозування результатів у гнучких середовищах розробки.

Методи дослідження включають аналіз наукових і технічних джерел для вивчення сучасного стану проблеми, порівняльний аналіз підходів до моделювання, методи системної динаміки для побудови моделей процесів, експериментальні методи для перевірки ефективності моделей, а також статистичний аналіз отриманих результатів.

У роботі розглядається застосування методів моделювання для підвищення прозорості процесів розробки, оптимізації використання ресурсів та покращення якості управління проєктами в Agile-командах. Особлива увага приділяється аналізу взаємозв’язків між елементами процесу розробки, впливу людського фактора та використанню сучасних технологій для підтримки прийняття рішень. Очікується, що результати дослідження сприятимуть підвищенню ефективності розробки програмного забезпечення, зменшенню ризиків і покращенню прогнозованості результатів.

Бакалаврська робота містить 95 сторінок, 16 рисунків, 11 таблиць, 3 розділи, висновки та список використаних джерел із 40 найменувань.

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ПРОГРАМНИХ ПРОЦЕСІВ В AGILE-СЕРЕДОВИЩІ

1.1 Передумови розвитку Agile-методологій та огляд пов'язаних досліджень

Широке впровадження гнучких методологій та фреймворків розробки програмного забезпечення змінило спосіб його створення [1, 2]. Ці зміни пов'язані з акцентом гнучких методологій на постійній адаптації для задоволення потреб клієнта та на пріоритеті частоті поставки програмного забезпечення над вичерпною документацією. У певних галузях, таких як онлайн-роздрібна торгівля або ігри, гнучкі практики стали домінуючою практикою розробки програмного забезпечення [3]. Одним із пояснень впровадження гнучких методологій є те, що вони надають відповіді на давні проблеми традиційних підходів, такі як проекти, що тривають з плином часу або бюджету, та продукти, що не відповідають вимогам клієнтів, сприяючи тісній співпраці між зацікавленими сторонами. Таким чином, вони задовольняють бізнес-потреби в інноваціях у динамічних середовищах [4].

Agile не лише впроваджує нові практики, але й підкреслює простоту, опускаючи частини традиційної розробницької роботи [5]. Поки що незрозуміло, якою мірою концептуальне моделювання може стати жертвою таких рекомендацій. З одного боку, важливість концептуальних моделей, таких як діаграми класів або моделі бізнес-процесів для розробки програмного забезпечення, була підкреслена, часто з посиланням на традиційні підходи. Моделі бізнес-процесів, зокрема, розглядаються як ключовий інструмент для підтримки розробки вимог [10, 11], комунікації між зацікавленими сторонами [12] або проектування корпоративних систем [13]. Також були зроблені пропозиції щодо узгодження моделювання з Agile-практикою [14]. З іншого боку, зосередження уваги на робочому програмному забезпеченні [5] знецінює специфікацію та документацію, фази, на яких часто використовуються

концептуальні моделі [10]. Дійсно, цей акцент на коді резонує з дослідженнями, які ставлять під сумнів загальні переваги моделювання [15, 16].

У цій роботі ми розглядаємо дослідницьке питання, якою мірою моделі бізнес-процесів корисні для гнучкої розробки програмного забезпечення. Зокрема, ми провели багатометодне дослідження на основі систематичного огляду літератури (SLR), тематичного синтезу та фокус-групового дослідження з фахівцями-практиками. У нашому огляді визначено 37 відповідних статей, які разом повідомляють про 15 застосувань моделей бізнес-процесів у проектах гнучкої розробки програмного забезпечення на всіх етапах, зосереджуючись на системному аналізі та проектуванні. Ми перевіряємо ці способи використання за допомогою дизайну фокус-групового дослідження з фахівцями-практиками.

У цьому підрозділі описується передумови нашого дослідження. По-перше, ми знову розглянемо загальні концепції гнучкої розробки програмного забезпечення [6]. По-друге, ми обговоримо синергію між моделюванням та гнучкою розробкою програмного забезпечення.

Гнучка розробка програмного забезпечення (AGIL) – це загальний термін, що стосується різних підходів до розробки програмного забезпечення, побудованих на споріднених ідеях. Ці підходи мають спільну рису: розробка програмного забезпечення організована таким чином, щоб враховувати зміни та періодично визначати пріоритети вимог. Ці підходи також спираються на самоорганізовані та міжфункціональні команди, і вони дотримуються усталених практик, таких як парне програмування, розробка з тестуванням або планування функцій [17].

Основні ідеї цих практик узагальнено в Маніфесті гнучкої розробки програмного забезпечення [5], де визначено чотири цінності. По-перше, пріоритет надається окремим особам та взаємодіям над процесами та інструментами. По-друге, *пріоритет робочого* програмного забезпечення над комплексною документацією. По-третє, *пріоритет співпраці з клієнтами над переговорами щодо контракту*. По-четверте, і останнє, *реагування на зміни* має пріоритет над дотриманням плану. Слід зазначити, що ці чотири твердження

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

виражають пріоритети, а це означає, що аспекти, які втратили пріоритет, все ще заслуговують на увагу [5]. Аспекти, виділені курсивом, потім пріоритетні, коли потрібно приймати складні рішення. Ці цінності визначають суть того, що визначає гнучкий підхід до розробки програмного забезпечення.

Перша *цінність надає пріоритет окремим особам та взаємодії*. Таким чином, вона ставить під сумнів акцент традиційних підходів, таких як каскадний [18] та спіральний [19], на процесах та інструментах. У традиційних підходах люди повинні адаптувати себе та свої робочі практики до передумов, що забезпечуються жорстким процесом або інструментом [17]. На противагу цьому, гнучкі підходи визнають еволюціонуючу природу розробки програмного забезпечення як людської практики, яка може бути обмежена процесом або інструментом управління [20]. У гнучкій розробці програмного забезпечення зміни та непередбачувані події приймаються, а самоорганізовані команди є ключем до отримання кінцевого продукту хорошої якості [21]. З цієї причини покращення комунікації та адаптація процесів ставляться на перше місце [22].

Друга *цінність надає пріоритет робочому програмному забезпеченню*. Робоче програмне забезпечення є кінцевим і найважливішим продуктом, тоді як документація розглядається як абстракція, що підтримує цю кінцеву мету [16]. Беручи це до уваги, друга *цінність* вимагає обмеженого обсягу та деталізації документації залежно від потреб команди. Часта взаємодія та комунікація зменшують та замінюють потребу в документації [23]. На противагу цьому, традиційні підходи до розробки програмного забезпечення підкреслюють важливість детальної документації. Для багатьох практиків це вважається перевантаженням, і це не обов'язково допомагає в розумінні програм або задоволенні потреб клієнтів [24].

Третя *цінність надає пріоритет співпраці з клієнтами*. Представляють цю цінність як рішення для запобігання перетворенню контрактів на джерело збільшення часу, витрат та перешкод для змін. Альтернативою співпраці на основі контрактів є зосередження на постійній узгодженості з клієнтом протягом усього процесу розробки. У цьому контексті контракти використовуються для

визначення умов, що сприяють співпраці. У традиційних методологіях взаємодія з клієнтами є важливою для визначення вимог, але як тільки це робиться на етапі аналізу, їхня участь обмежується [25].

Четверта *цінність* закликає до прийняття змін. Прийняття змін є одним із найпоширеніших аспектів гнучких методологій [23, 17]. Ідея постійних змін ставить під сумнів традиційні підходи, які базуються на жорсткому плануванні [26]. Однак гнучкий підхід не виключає планування. Плани розробляються, щоб дати команді уявлення про те, що робити, та допомогти їм пом'якшити потенційні ризики. Ці плани можна адаптувати [23]. У той час як традиційні підходи визначають довгострокові плани, гнучкі підходи зосереджені на плануванні кількох коротких циклів розробки тривалістю не більше двох-чотирьох тижнів. Ці цикли називаються ітераціями. План для кожної ітерації розробляється на основі результатів попередньої ітерації, відгуків від усіх зацікавлених сторін, самоаналізу та спільних командних рішень [23].

Підсумовуючи, гнучкі підходи вирішують «проблеми, що характеризуються змінами, швидкістю та турбулентністю» проектів розробки програмного забезпечення [17]. Ці три характеристики є поширеними в сценаріях, де технології, бізнес-моделі та продукти швидко розвиваються. На відміну від традиційних методологій, гнучкі методології описують практики, засновані на цінностях, які підтримують ці сценарії. Ці цінності призводять до меншої документації, меншої бюрократії, більш інтенсивної співпраці, швидших циклів доставки, більшої адаптивності [25] та більшої ймовірності задоволення потреб клієнтів.

Концептуальне моделювання є ключовою темою в програмній інженерії [27, 10]. У цьому контексті модель – це абстрактне представлення, яке служить певній меті [28]. У проектах розробки програмного забезпечення моделювання допомагає розробнику зрозуміти дизайн, надаючи огляд та інструмент для спілкування з іншими розробниками та іншими зацікавленими сторонами [29]. Переваги моделей для підтримки системного аналізу та проектування були підкреслені щодо їхньої когнітивної ефективності [6].

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

Також обговорювалися переваги концептуальних моделей для гнучких підходів. На перший погляд, це може здатися нелогічним, оскільки моделі часто використовуються для документації [10, 16], а гнучкі підходи надають пріоритет іншим аспектам [30]. Однак було розроблено різні пропозиції щодо інтеграції моделювання та гнучких практик. Попередником інтеграції моделювання та гнучкого підходу є Амблер та його пропозиція щодо гнучкої розробки на основі моделей [31]. Ця пропозиція використовує високорівневі архітектурні моделі та моделі вимог як орієнтир. У кожній ітерації присвячується час моделюванню, а створені моделі надаються зацікавленим сторонам. Ми обговорюємо два систематичні огляди літератури щодо інтеграції гнучких методів та моделей по черзі. По-перше, Альфраї та Лано аналізують підходи до інтеграції гнучких методів та моделювання [14]. Їхній систематичний огляд літератури визначає спільні характеристики, переваги та проблеми цих різних підходів. Вони виявили, що підходи, що інтегрують гнучку та модельно-орієнтовану розробку, підкреслюють необхідність зробити моделювання простішим та швидшим. Головною метою інтеграції є підвищення продуктивності, що, згідно з їхнім оглядом, загалом досягнуто. Однак вони також наголошують, що більшість підходів, що інтегрують гнучку та модельно-орієнтовану розробку, є відносно новими. Потрібні подальші дослідження для вивчення їхньої ефективності.

По-друге, зосереджуючись на сучасному стані поєднання гнучких методологій та моделювання в цілому. Їхній систематичний огляд літератури - підкреслює, що коли гнучкі та модельно-орієнтовані підходи до розробки використовуються незалежно, вони не відповідають потребам складних програмних проєктів [7]. Їх інтеграція дозволяє скоротити час, необхідний для проєкту, збільшити залучення зацікавлених сторін, фіксувати зміни архітектури, покращити якість та полегшити валідацію. Зазначають, що більшість підходів пропонують гнучкі альтернативи для моделювання, а не оцінюють, як підтримувати гнучкі методи за допомогою модельно-орієнтованої розробки [8]. Прикладом останнього є пропозиція, яка включає, серед іншого, парне моделювання, ітеративне та інкрементне моделювання, щоденне оновлення

плану модельно-орієнтованої розробки та план модельно-орієнтованої розробки для наступного спринту .

Обидва огляди літератури роблять висновок, що дослідження інтеграції гнучких методологій та моделювання все ще потребують подальшого вивчення. Зокрема, необхідні реальні програми, щоб повноцінно зрозуміти та використати переваги інтеграції гнучких методологій з моделюванням. Помітною прогалиною в дослідженнях у цьому контексті є той факт, що ці огляди в основному зосереджувалися на моделях UML. Це важливе упущення, оскільки моделі бізнес-процесів є важливими для розробки вимог та їх узгодження з потребами бізнесу [10]. Різні підходи поєднують моделі бізнес-процесів та гнучку розробку програмного забезпечення. Досі дослідження з цієї теми залишаються фрагментарними, і систематичний огляд інтеграції моделей бізнес-процесів та гнучкої розробки програмного забезпечення поки що відсутній. Наше дослідження має на меті заповнити цю прогалину.

1.2 Основні підходи до моделювання програмних процесів

Метою цього дослідження є огляд використання та категорій застосування моделювання бізнес-процесів у гнучких проектах розробки програмного забезпечення. З цією метою ми дотримуємося багатометодного підходу. Більш конкретно, ми спочатку проводимо систематичний огляд літератури (SLR). У цьому контексті ми вибираємо дослідження, що поєднують моделі бізнес-процесів та гнучку розробку програмного забезпечення. По-друге, ми проводимо тематичний аналіз вибраних досліджень на основі кроків. Цей тематичний аналіз базується на процедурах якісного кодування [37] для визначення використання моделей бізнес-процесів у гнучкій розробці програмного забезпечення. Таким чином, ми також визначаємо фази циклу розробки програмного забезпечення, де застосовуються ці використання. По-третє і останнє, ми проводимо фокус-групове дослідження з практиками, дотримуючись рекомендацій. Фокус-група оцінює, на якому рівні виявлені

використання можуть підтримувати гнучку розробку програмного забезпечення у чотирьох категорія: управління, команда, технології та процеси.

Далі, у розділі представлені дослідницькі питання, що лежать в основі нашого дослідження, та їх зв'язок із застосованими методами. Пояснюється процедура відбору статей та вилучення загальних даних. Описано, як ми ідентифікуємо та кодуємо використання моделей бізнес-процесів у гнучкій розробці програмного забезпечення, детально описано підготовку та проведення фокус-групи.

Ми дотримувалися рекомендацій щодо проведення систематичних - оглядів літератури з програмної інженерії. З цієї причини ми чітко визначаємо дослідницькі питання, які визначають цілі цього дослідження.

RQ1: Яке використання бізнес-процесних моделей у гнучких проектах розробки програмного забезпечення? Процесні моделі використовувалися по-різному в гнучких проектах, і мета полягає у визначенні цих різних застосувань

RQ2: На яких фазах гнучкого проєкту розробки програмного забезпечення використовуються моделі бізнес-процесів? Гнучкий проєкт розробки програмного забезпечення складається з різних фаз. Ми хочемо визначити, на яких фазах і як використовуються моделі бізнес-процесів.

ЗПЗ: Яка категоризація цих застосувань відповідно до управлінського, командного, технологічного та процесного підходу? Ми аналізуємо стосовно того, який із чотирьох аспектів використовуються моделі бізнес-процесів.

Щоб відповісти на ці дослідницькі питання, ми діятимемо, як узагальнено на рисунку 1.1. На рисунку показано загальний процес разом із вхідними та вихідними даними.

Ми обговорюємо по черзі процес пошуку, вибір дослідження та вилучення даних.

Процес відбору статей показано в таблиці 1.1. Пошук проводився з 25 по 28 травня 2020 року у двох онлайн-дослідницьких базах даних: Web of Science (WoS) та Google Scholar.

Було використано два набори комбінацій ключових слів, один для WoS, а

інший для Scholar. Обидва вони базувалися на трьох пошукових термінах: *використання, моделювання бізнес-процесів та гнучка розробка програмного забезпечення*. У WoS було додано блок синонімів та пов'язаних термінів до кожного з цих ключових слів, як описано в таблиці 1.2.

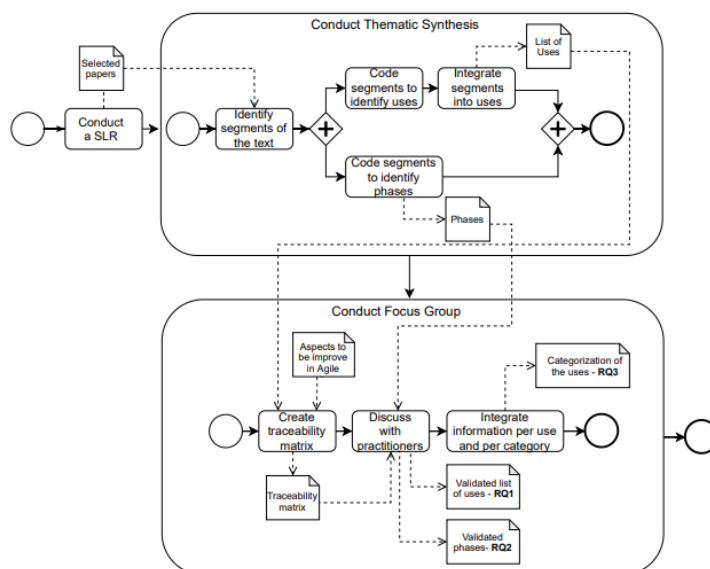


Рисунок 1.1 - Модель процесу, що відображає взаємодію між методами та дослідницькими питаннями

У Scholar ми використовували комбінації ключових слів, пов'язаних з «*моделюванням бізнес-процесів*» та «*гнучкою розробкою програмного забезпечення*». Цей пошук дав близько 149 520 результатів, і перші 270 результатів були розглянуті в цьому дослідженні. Причиною цього граничного значення було зниження релевантності досліджень, які з'являлися далі в результатах пошуку Scholar. Пошук у дослідницьких базах даних призвів до 441 дослідження. Для 441 дослідження з вищезазначеного пошуку ми виконали кроки відбору, як зазначено в Таблиці 1.1. Релевантність кожного визначеного дослідження спочатку перевірялася на основі назви, а потім на основі анотації, що призвело до видалення 324 первинних досліджень. З решти 84 досліджень ми спочатку видалили дублікати, а потім прочитали їх усі повністю. Під час відбору застосовувалися такі критерії включення. Стаття повинна бути написана

англійською мовою (C1), мати відношення до гнучкої розробки програмного забезпечення (C2) та представляти зв'язок між моделями бізнес-процесів та гнучкою розробкою програмного забезпечення (C3).

Таблиця 1.1:

Процес пошуку та відбору (SLR)

Крок	Опис	К-сть статей WoS	К-сть статей Scholar	Всього
1	Виконати пошук за назвою відповідно до ключових слів	171	270	441
2	Прочитати всі назви	86	77	163
3	Прочитати анотації (abstracts)	31	53	84
4	Видалити дублікати та застосувати критерії відбору	13	28	41
5	Застосувати критерії оцінки якості	13	24	37

Як показано в Таблиці 1.1, ми використовували критерії оцінки якості Кітченхема та ін. [40]. Вони визначаються трьома питаннями: по-перше, чи описані та належним чином висвітлені наші критерії відбору? По-друге, чи оцінили дослідники валідність свого дослідження? По-третє, чи було дослідження описано адекватно? Для кожного критерію оцінки існують три можливі бали залежно від застосовності: повний (1 бал), частковий (0,5 бала) та ні (0 балів). Сума балів за різними критеріями на кожному дослідницьку роботу називається балом оцінки якості. Ми вирішили включити всі дослідження з балом щонайменше 1, що дало 37 первинних досліджень [9].

Дані, які ми витягли з кожної статті, включали бібліографічні дані (назва, рік публікації та джерело) та контекстні дані (застосовна область, методологія гнучкої розробки програмного забезпечення, мова моделювання, оцінка якості, метод валідації та тип продукту). Ці дані були зібрані для кращого розуміння вибраних досліджень і витягнуті у вигляді атрибутів або описових міток [37] для полегшення аналізу. Метод валідації був класифікований відповідно до експериментальних моделей для валідації в інформатиці.

Після вибору первинних досліджень та вилучення даних, пов'язаних з

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

роботами, ми визначаємо використання моделей бізнес-процесів у гнучкій розробці програмного забезпечення, як представлено в статтях. Це робиться з метою відповіді на RQ1. З цією метою ми використовуємо кроки, як посилання для тематичного синтезу. Спочатку ми повністю читаємо кожну з вибраних статей та визначаємо конкретні сегменти тексту, що описують взаємодію між моделями бізнес-процесів та гнучкою розробкою. Загалом у 37 дослідженнях було визначено 59 сегментів (S). Ми кодували сегменти, використовуючи індуктивний підхід [36] та процедури кодування. Це робиться з урахуванням RQ1 та контексту кожної статті [10].

Таблиця 1.2:

Ключові слова для пошуку в Web of Science

Тип	Блок 1	Блок 2	Блок 3
Синоніми	AND (use OR application adoption)	AND ("business process modeling" OR BPM)	AND ("agile software development" OR "agile development")
Споріднені терміни	AND (benefit OR need OR help OR value OR advantage OR approach)	AND (model OR "business process management")	AND (SCRUM OR kanban OR XP OR "extreme programming" OR "agile methodologies" OR "agile software development methodologies")
Ширші терміни	-	AND(modeling)	AND (agile)

Другий цикл кодування відповідає пропозиціям, щоб зменшити перекриття та інтегрувати сегменти (S) у змістовні представлення. З цією метою ми розглянули як індуктивний підхід [36], так і процедуру шаблонів [37]. Обидві процедури кодування виконуються одним членом авторської команди та спочатку перевіряються одним практиком. Різні думки обговорювалися, доки кодер та перевіряльник не досягли згоди. В результаті ми визначаємо набір застосувань моделей бізнес-процесів у гнучкій розробці програмного забезпечення (U). Кожне використання позначено як U_p , де $p \in N$ та $U_p \in U$. Ці способи використання та відповідні вихідні дані знову перевіряються в рамках діяльності фокус-групи з фахівцями-практиками.

Щоб відповісти на RQ2, ми застосували аналогічні процедури кодування. Для кожного сегмента ми визначаємо фазу розробки програмного забезпечення, на якій використовуються моделі бізнес-процесів. Це досягається шляхом використання визначень життєвого циклу розробки системи та сегмента в контексті кожної дослідницької роботи як орієнтир.

Тепер, коли ми визначили використання (U), ми провели фокус-групове дослідження з фахівцями відповідно до рекомендацій Моргана [38]. Метою фокус-групи є вирішення питання RQ3, яке має на меті визначити, на якому рівні визначені використання можуть підтримувати гнучку розробку програмного забезпечення у чотирьох сферах: управління, команда, технології та процеси. Основними вхідними даними є дані, пов'язані з використаннями, визначеними в дослідницькій літературі (щодо RQ1). Ми провели другу перевірку інтеграції цих застосувань під час першого засідання фокус-групи [11]. Кодування фаз розробки програмного забезпечення, пов'язаних з використанням, також було перевірено з фахівцями.

Обговорення проводилися у трьох онлайн-зустрічах між 18 та 20 липня 2020 року. Щодня проводилася одна зустріч, і всі вони тривали 9 годин 30 хвилин. На всіх зустрічах були присутні ті самі фахівці.

Для фокус-групових сесій ми підготували матрицю відстеження вимог. Ця матриця є поширеним інструментом, який використовується для демонстрації зв'язку між тим, що потрібно (тобто вимогами), та якими артефактами це буде задоволено (наприклад, функціональністю, варіантами використання, діями). Ми створили матрицю відстеження $T = (R, S, X)$ на основі рекомендацій, де R , S та X описані наступним чином:

- Вимоги (R) – це аспекти, які потребують підтримки в гнучкій розробці програмного забезпечення. Кожен аспект позначається як R_n , де $n \in \mathbb{N}$: $n < 25$, та $R_n \in R$. Ці вимоги класифікуються на чотири категорії (C) аспектів (MN для управління, TM для команди, TH для технологій, PR для процесу), які описані в таблиці 1.3.

- Сегменти (S) описують, як моделі бізнес-процесів використовуються

для підтримки завдань Agile. Кожен сегмент називається S_m , де $m \in \mathbb{N}: m < 59$, та $S_m \in S$.

Таблиця 1.3:

Опис категорій (Менеджмент, Команда, Технології, Процес)

Категорія	Опис
Менеджмент	(R1) Визначати цілі та стратегії, (R2) приймати рішення, (R3) управляти культурними відмінностями, (R4) залучати стейкхолдерів, (R5) створювати ефективну комунікацію та (R6) управляти різницею в часі.
Команда	(R7) Створювати взаємодію, (R8) обмінюватися знаннями, (R9) управляти розміром команди, (R10) управляти залученням клієнта, (R11) управляти недосвідченими розробниками, (R12) покращувати розподіл завдань, (R13) отримувати частий зворотний зв'язок та (R14) покращувати відносини між клієнтами та постачальниками.
Технології	(R15) Покращувати технологічне навчання, (R16) підтримка технологічних ресурсів, (R17) контроль бюджету та (R18) підтримка комунікаційних технологій.
Процес	(R19) Визначати потік інформації, (R20) визначати обсяг проєкту, (R21) підтримувати проєктну документацію, (R22) підтримувати механізм оцінки agile-процесів, (R23) підтримувати специфікацію вимог, (R24) визначати ролі та обов'язки та (R25) підтримувати складні проєкти.

Зв'язок між вимогою R_n та сегментом S_m представлений знаком «х» на перетині цих двох компонентів. Існування зв'язку означає, що використання моделей бізнес-процесів, описаних у сегменті S_m , може підтримувати складний аспект R_n гнучкої розробки програмного забезпечення. Набір зв'язків між вимогою R та сегментами S позначається як $XCR \times S$.

Коли сегмент S_m , витягнутий з вибраних досліджень, чітко описує та відповідає одному або кільком аспектам R_n категоризації, представленої в Таблиці 3, ми реєстрували це відношення в матриці простежуваності T . Крім того, враховуючи контекст кожного з вибраних досліджень та описи, представлені в Таблиці 1.3, ми реєстрували додаткові можливі застосування для кожного із сегментів. Потім фахівці перевіряють цю матрицю простежуваності в обговореннях фокус-групи. Загальна структура цієї матриці зображена на Рисунок 1.2.

Вважалося за доцільне для цього дослідження залучити учасників зі значним досвідом у проєктах розробки програмного забезпечення загалом та

зокрема у гнучкій розробці програмного забезпечення. Усі учасники були знайомі з різними методологіями гнучкої розробки програмного забезпечення, а також з традиційними підходами [12]. Прагнучи отримати широкий спектр сприйняття та досвіду, а також зменшити упередженість, ми відібрали фахівців з різними ролями в індустрії програмного забезпечення.

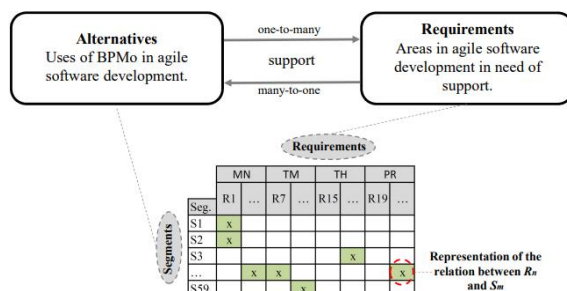


Рисунок 1.2 - Структура матриці простежуваності

Чотири фахівці (менеджер ІТ-проектів, архітектор програмного забезпечення, старший розробник, спеціаліст з інженерії вимог) взяли участь у фокус-групових обговореннях (див. табл. 1.4). Вони мають досвід роботи в ІТ-індустрії від 10 до 25 років та від 6 до 10 років у гнучкій розробці програмного забезпечення. На момент проведення цього дослідження всі вони працювали в ІТ-проектах багатонаціональних компаній, що спеціалізуються на електронній комерції (учасник 1), мобільності та фінансових послугах (учасник 2), автомобільній промисловості (учасник 3) та розробці програмного забезпечення (учасник 4). Для керівництва обговоренням та врахування динаміки групи на сесіях також був присутній фасилітатор.

Процес фокус-групи складається з двох фаз: кодування та перевірки матриці відстежуваності. Вхідними даними для першої фази є сегменти (S) та відповідне кодування, проведене в процесі тематичного синтезу.

Таблиця 1.4:

Учасники фокус-групи

#	Поточна роль	Попередні ролі	Років досвіду	Років досвіду
---	--------------	----------------	---------------	---------------

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

			в IT	з Agile
1	IT-проектний менеджер	Керівник РМО, Product Owner, бізнес-аналітик	25	10
2	Старший інженер з вимог	IT-консультант, бізнес-аналітик	10	7
3	Старший архітектор ПЗ	IT-координатор, техлід, інженер ПЗ, старший розробник	15	7
4	Старший розробник	Техлід, розробник	14	6

Під час перевірки коду ми отримуємо перевірений список застосувань моделей бізнес-процесів у гнучкій розробці програмного забезпечення (U) та пов'язані з ними фази. На цьому етапі фасилітатор представляє кожен сегмент S_m , який належить до кожного інтегрованого використання U_p . Потім кожен учасник перевіряє код. Це дає інтегрований список застосувань. Зрештою, учасники обговорюють потенційні розбіжності, доки не досягнуть консенсусу.

На другому етапі фахівці оцінюють зв'язки в матриці відстеження за допомогою трьох кроків. По-перше, для кожного сегмента S_m матриці відстеження фасилітатор представляє учасникам контекст відповідного документа. По-друге, представляють зв'язки між кожним сегментом S_m та аспектами R_n , пов'язаними з чотирма категоріями (менеджмент, команда, технологія та процес). По-третє, учасники обговорюють виявлені та можливі зв'язки. На основі цього обговорення учасники голосують за збереження, видалення або додавання зв'язку між сегментом та вимогами.

Загальна мета аналізу полягає в отриманні перевіреного списку застосувань (U) та визначенні, на якому рівні ідентифіковані використання можуть підтримувати гнучку розробку програмного забезпечення. З цією метою матриця відстеження T фіксує, як моделі бізнес-процесів можуть підтримувати різні аспекти гнучкої розробки програмного забезпечення відповідно до того, що зазначено в літературі. Питання полягає в тому, чи можна інтегрувати сегменти та зв'язки. З цією метою для кожного використання U_p у кожній Категорії ми підраховали кількість зв'язків на S_m у Категорії (Управління, Команда, Технологія, Процес) [13]. Потім ми зважили результат відповідно до кількості аспектів (R_n), які належать до Категорії. Далі ми підсумували значення,

розраховані для кожного S_m , що належить до U_p . Нарешті, ми поділили значення суми на кількість S_m , що належать до U_p . Результат цієї процедури представлено в розділі як частину RQ3,

1.3 Припущення та принципи, що лежать в основі гнучких процесів розробки

В останні роки в літературі було запропоновано низку процесів, які претендують на звання «агільних». Щоб уникнути плутанини щодо того, що означає термін «агільні», сімнадцять методологів та прихильників агільних процесів зустрілися, щоб обговорити та дійти згоди щодо того, що означає «агільність». Результатом зустрічі стало створення Agile Alliance та публікація маніфесту, який містив перелік принципів, які повинні підтримувати агільні процеси [1]. Короткий виклад цих принципів (пронумерованих та в порядку, як повідомляє Agile Alliance [1]) наведено в таблиці нижче.

Маніфест «Agile Alliance» – це стисло визначення цінностей та цілей «Agile Software Development» (Гнучка розробка програмного забезпечення) та детально описаний через ці принципи, які можна розглядати як набір політик та правил, що повинні підтримуватися процесами, що претендують на роль «гнучких» [14]. Ці принципи забезпечують гарну основу для визначення припущень, що лежать в основі гнучких процесів. У наступному розділі ми розглянемо ці принципи та визначимо припущення, які, як видається, робляться під час прийняття цих принципів.

На рисунку 1.3 підсумовано наше бачення взаємозв'язку між Принципами, Практиками, Припущеннями та Обмеженнями. Ми дотримуємося думки, що існують припущення, зазвичай невисловлені, які призвели до прийняття принципів Agile Alliance. Існують також припущення розробників (знову ж таки невисловлені) щодо значення цих принципів та їхньої відносної важливості. На основі цих принципів та припущень встановлюються практики розробки. Навмисно чи ні, ці припущення призводять до обмежень у

результуючих гнучких процесах. Обмеження, що обговорюються далі в цій статті, частково ґрунтуються на нашій оцінці припущень, що лежать в основі принципів та практик гнучких процесів. У цьому розділі ми визначаємо припущення, що лежать в основі принципів Agile Alliance, як ми їх сприймаємо [15]. Обговорення в цьому розділі організовано навколо кластерів пов'язаних принципів, де кожен кластер породжує окремий набір припущень. Ми також визначаємо приклади ситуацій, у яких припущення можуть не виконуватися.

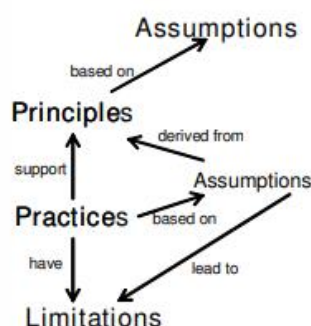


Рисунок 1.3 - Взаємозв'язки між принципами, практиками, припущеннями та обмеженнями

Принцип 1. «Нашим найвищим пріоритетом є задоволення потреб клієнта шляхом раннього та безперервного постачання цінного програмного забезпечення».

Принцип 4. «Часто постачайте робоче програмне забезпечення».

Принцип 5. «Працююче програмне забезпечення є основним показником прогресу».

Важливість Принципу 1 нагадує розробникам, що програмне забезпечення розробляється для виконання послуг, які додають цінності користувачам під час їх поставки. (Принципи 4 та 5 можна розглядати як наслідки принципу 1, і тому вони обговорюються тут разом.) Розробникам та планувальникам проектів необхідно пам'ятати, що потреби клієнтів розвиваються в процесі використання систем. Підтримка Принципу 4 дозволяє розробникам оцінювати та задовольняти потреби клієнтів, що змінюються. Гнучкі процеси підтримують ці принципи, структуруючи діяльність з розробки

на короткі ітерації з фіксованим часом, кожна з яких створює робочий код. Ці ітерації з фіксованим часом роблять гнучкі процеси передбачуваними в часі. Ціна полягає в тому, що обсяг продукту може бути непередбачуваним: дотримуючись терміну ітерації, розробники можуть вирішити не впроваджувати функції, спочатку позначені для реалізації в ітерації. Гнучкі процеси використовують практики, які допомагають розробникам мінімізувати час, необхідний для реалізації виявлених вимог у робочому коді [16]. У XP ці практики включають створення простих проектів, безперервну інтеграцію, колективне володіння кодом та рефакторинг.

Часта поставка робочого коду забезпечує прозорість проекту з точки зору, яку клієнт може зрозуміти (тобто з точки зору виконуваного продукту, що розвивається), а не з точки зору планів, що розвиваються, у формі вимог та проектної документації, які часто не представлені в термінах, які клієнт може зрозуміти або зрозуміти. Клієнти можуть використовувати прирости, що надаються ітераціями, як основу для (1) визначення прогресу проекту, як зазначено в Принципі 5, та (2) уточнення та вдосконалення вимог. Короткі ітерації сприяють своєчасному зворотному зв'язку з клієнтами, що може допомогти гарантувати, що кінцевий продукт відповідатиме потребам клієнта на момент поставки.

Важливість залучення кінцевих користувачів та клієнтів до процесу розробки програмного забезпечення широко визнана та була основною мотивацією роботи на початку 1980-х років над розробкою програмного забезпечення для «кінцевого користувача» [12]. Значна частина цієї ранньої роботи була зосереджена на розробці механізмів, які дозволили б кінцевим користувачам безпосередньо брати участь у розробці вимог та проектів, а також розуміти артефакти, створені розробниками програмного забезпечення. Робота в той час була зосереджена на розробці вимог та проектних нотацій, які є «зручними для клієнта», тобто які можна використовувати для створення специфікацій вимог та проектів, що надають клієнтам суттєве розуміння вимог та проектів. Такі механізми, як мови програмування четвертого покоління (4GL)

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

та виконувані вимоги [4] [30] [35] [36] та проектні специфікації, вважалися технологіями, що сприяють розробці програмного забезпечення для кінцевих користувачів. Замість того, щоб наголошувати на технічних факторах взаємодії з клієнтами, гнучкі процеси роблять акцент на постійній співпраці команд розробників та клієнтів.

Припущення щодо видимості: Видимість проекту може бути досягнута виключно шляхом надання робочого коду [17].

Видимість проектів розробки програмного забезпечення традиційно досягається за допомогою звітів, специфікацій та показників якості й продуктивності, а робочий додаток (код) видно лише після того, як розробник виконав великий обсяг роботи та витратив багато часу на роботу над проектом. Клієнтам легше зрозуміти, чи просувається проект у потрібному напрямку, якщо вони можуть реально *побачити* інтерфейс користувача та те, як програмне забезпечення *виконує* необхідні їм завдання, а не просто покладатися на звіти, специфікації та інші показники. Оскільки клієнта дійсно цікавить кінцевий продукт, основним показником прогресу в гнучких процесах є код, розроблений у *проекті*.

Це добре працює для програмного забезпечення, оснащеного користувацькими інтерфейсами, що розвиваються з часом. У проектах, в яких користувацький інтерфейс не завжди є частиною кінцевого продукту, або в проектах, пов'язаних з розробкою систем без людського інтерфейсу (наприклад, вбудовані системи), потрібні інші засоби видимості. Для таких проектів певну видимість можуть забезпечити симуляції програмного забезпечення, охоплення приймальних тестів, а також формальні огляди та перевірки кінцевих продуктів.

Припущення щодо ітерації : Проект завжди можна структурувати на короткі ітерації з фіксованим часом.

Гнучкі процеси вимагають від розробників групування необхідних функцій у слабопов'язані пакети, кожен з яких можна реалізувати за короткі ітерації з фіксованим часом. Така декомпозиція зазвичай базується на неявно нав'язаній архітектурі, що складається з слабопов'язаних модулів, і тому є

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

бажаною. Тут припускається, що розроблюваний застосунок можна розбити на невеликі дискретні інкременти, які можна розробляти та демонструвати за короткі фіксовані інтервали часу, і що після кожної з цих ітерацій клієнт зможе спостерігати додаткову функціональність у продукті [18]. Завдяки цьому клієнт зможе часто надавати зворотний зв'язок щодо прогресу проекту, вказуючи, чи розробляється він так, як потрібно/очікується, чи ні.

Структурування роботи за невеликими пакетами, які можна швидко впровадити, не завжди можливо. Наприклад, у деяких складних системах застосунку може знадобитися взаємодія з низкою підсистем складним чином лише для надання базових послуг, цінних для клієнтів. У таких ситуаціях може бути неможливо створити достатньо малі пакети функцій для виконання за одну ітерацію через тісні залежності.

Принцип 2. «Бізнесмени та розробники повинні щодня працювати разом протягом усього проєкту». Взаємодія між розробниками та кінцевими користувачами в гнучких процесах пов'язана, головним чином, з вирішенням проблем, пов'язаних з функціями, та визначенням обсягу робіт. Ця взаємодія відбувається не лише на початку проєкту; вона відбувається протягом усього процесу. Зокрема, гнучкі процеси пропагують взаємодію, яка може включати (1) надання клієнтами інформації у формі неформальних описів очікуваної поведінки (наприклад, історій в екстремальному програмуванні), (2) відповіді на запитання про бажані функції, (3) співпрацю з розробниками у вирішенні проблем, що стосуються функцій, що мають бути впроваджені, та (4) співпрацю з розробниками для розробки планів проєкту.

Одним із найважливіших наслідків такої тісної співпраці між користувачами та розробниками є краще розуміння проблем і потреб одне одного, що зменшує проблеми людської взаємодії та таким чином значно підвищує шанси на успішний результат проєкту.

Цей принцип застосовується не лише до взаємодії між розробниками та клієнтами; він також поширюється на взаємодію між розробниками. Часта взаємодія дозволяє розробникам швидко вирішувати проблеми та

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

непорозуміння, а також швидше та надійніше просуватися вперед над проектом.

Припущення щодо взаємодії з клієнтами : команди клієнтів доступні для частоті взаємодії, коли це необхідно розробникам.

Деякі основні припущення тут полягають у тому, що клієнт доступний у той час, коли розробникам потрібно з ним взаємодіяти, і що клієнт завжди може перенести іншу роботу, щоб у нього був час для частоті взаємодії з розробниками. Реальність така, що клієнт не завжди може перенести іншу роботу.

Припущення щодо командної комунікації: Розробники розташовані в часі та місці таким чином, що вони можуть часто та інтенсивно спілкуватися один з одним.

Це припущення дуже схоже на припущення про взаємодію з клієнтом, але зосереджене на здатності розробників взаємодіяти один з одним. Так само, як припущення про взаємодію з клієнтом передбачає певну спільну кількість часу, місця, ресурсів та доступності, так само діє і припущення про командну комунікацію. Час, місце, ресурси та доступність мають бути скоординовані та забезпечені для того, щоб цей принцип міг підтримуватися. Існує безліч прикладів проектів, у яких це припущення не виконується. Нерідко команди розробників, що працюють над одним проектом, розосереджені по широких географічних областях, що охоплюють багато часових поясів.

Принцип 8. «Найефективніший та найдієвіший метод передачі інформації команді розробників та всередині неї – це особиста розмова». У гнучких процесах наголос робиться на особистому спілкуванні, а не на формальній та точній документації, а також на теле-/відеоконференціях чи розмовах електронною поштою. Спільнота гнучких процесів стверджує, що неформальне особисте спілкування дає більше користі, ніж спілкування на основі формальної документації, навіть якщо можливість відстежувати всю інформацію зникає [19]. Перевагою особистого спілкування є те, що залучені сторони можуть змінювати напрямок обговорення за потреби, щоб отримати розуміння обговорюваної теми, а також можуть спостерігати та реагувати на невербальні сигнали комунікації під час взаємодії розробників та/або клієнтів.

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		28

Навіть якщо формальні та технічні механізми комунікації не заохочуються, слід використовувати протоколи або списки «справ» для відстеження обговорюваних питань.

Припущення щодо особистої взаємодії: Особиста взаємодія є найпродуктивнішим методом спілкування з клієнтами та між розробниками.

Важко уявити реалізацію Принципу 8 без спільного розташування клієнтів та розробників і без графіків, які дозволяють часту взаємодію під час проекту. Якщо клієнти та розробники знаходяться разом, навіть до такої міри, як вони можуть пройти коридором і поговорити один з одним, то будь-які питання, проблеми чи труднощі можна негайно вирішити, і чітке рішення можна буде негайно отримати. Без особистого контакту зростає потенціал для непорозумінь, і завжди існує складність встановлення контакту – проблема «телефонної мітки» або «мітки електронної пошти».

Зменшення акценту на документації як засобі комунікації ґрунтується на припущенні, що неявні знання мають бути ціннішими за екстерналізовані. Прихильники зазначають, що розробникам необхідно інтерналізувати екстерналізовані знання, щоб зробити їх корисними, і що навчання може бути досягнуто шляхом обміну неявними знаннями через розмови [27]. Критики стверджують, що зосередження на неявних знаннях робить проекти, які використовують гнучкі процеси, залежними від експертів [10]. Ще одне занепокоєння полягає в тому, що цінування неявних знань над екстерналізованими може призвести до втрати корпоративної пам'яті та зниження здатності організації системно навчатися на своєму колективному досвіді [20]. Організація, яка стурбована своєю здатністю ефективно навчатися на минулому досвіді, повинна цінувати як неявні, так і екстерналізовані знання та розуміти їхню взаємодію. Неявні знання мають вирішальне значення для побудови екстерналізованих знань, як зазначає Нонака [25][37], а екстерналізовані знання можуть взаємодіяти з неявними знаннями, щоб виявити приховані або створити нові неявні знання. Організації, які цінують системне навчання, повинні сприяти середовищу, яке не лише підтримує обмін неявними

знаннями, але й підтримує екстерналізацію неявних знань.

Припущення щодо документації : Розробка обширної (відносно повної) та узгодженої документації й моделей програмного забезпечення є контрпродуктивною.

З огляду на зосередженість Agile-розробників на коді (див. принципи 1, 4 та 5 вище), таке применшення значення документації та моделей програмного забезпечення не дивує. Припускається, що визначення специфікацій та дизайну з коду є надійнішим, ніж з інших документів, особливо враховуючи, що специфікації, вимоги, проектна документація та моделі можуть не оновлюватися після зміни коду. Таким чином, код є найточнішим та найнадійнішим описом того, що робить система та як вона була розроблена.

Причиною розчарування спільноти гнучких процесів моделюванням може бути попередній досвід роботи з комерційними інструментами моделювання, які були не що інше, як спеціально розроблені середовища малювання. Такі інструменти надавали дуже мало підтримки для складніших завдань підтримки (1) зв'язків відстеження між моделями та (2) узгодженості між моделями та їх реалізаціями. Сучасні інструменти моделювання дещо вийшли за рамки цього і тепер забезпечують підтримку генерації коду та циклічної інженерії. Що ще важливіше, основні постачальники інструментів зараз розширюють свої пропозиції для підтримки підходу до розробки програмного забезпечення на основі моделей Object Management Group, відомого як Модельно-орієнтована архітектура (MDA) [28]. MDA базується на розділенні платформи-орієнтованих системних деталей від платформи-незалежних деталей. Інструменти на основі MDA надають механізми для зіставлення платформи-незалежних деталей з платформи-специфічними реалізаціями, причому значна частина зіставлення автоматизована за допомогою шаблонів, шаблонів та інших форм повторно використовуваних інтерфейсів. У підході MDA моделі є центральними артефактами, а використання інструментів MDA може допомогти пришвидшити розробку завдяки автоматизованій генерації значних частин коду програми та проміжного програмного забезпечення, а також підвищенню рівня

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

абстракції, на якому працюють розробники.

Для клієнтів, які укладають контракти з розробниками на надання систем, точні моделі можуть бути не обов'язковими. Однак, існують ситуації, коли моделі є цінними самі по собі, і коли було б корисно зберегти ці моделі для подальшого використання. Деякі з цих ситуацій описані нижче:

- Розвиток великих складних систем з тривалим життєвим циклом: наявність хороших моделей може зменшити витрати та зусилля, пов'язані з модифікацією таких систем. Без цих моделей розробники змушені аналізувати вихідний код, щоб зрозуміти його та визначити вплив змін. Дослідження показали, що значна частина зусиль, необхідних для розвитку систем, витрачається на розуміння коду. Хороші моделі можуть допомогти полегшити це завдання.

- Управління корпоративними системами для забезпечення відповідності бізнес-цілям: Хороші моделі бізнес-процесів та систем можуть використовуватися корпоративними архітекторами для (1) перевірки відповідності запланованих та впроваджених систем бізнес-цілям, (2) визначення того, як можна компонувати існуючі системні сервіси для створення нових сервісів, (3) виявлення надлишків у системах (особливо коли організації або підрганізації зливаються з іншими організаціями або підрганізаціями), (4) визначення досвіду розробки, який можна використовувати повторно, та (5) визначення впливу змін на існуючі системи. Моделі бізнес- та системного рівня, чітко визначені зіставлення між ними та відповідність коду е можуть значно покращити управління корпоративними системами.

Хороші моделі та документація також можуть бути використані для того, щоб нові співробітники швидко ознайомилися з бізнесом та системами, що розробляються, а також щоб допомогти користувачам компонентів визначити, чи дійсно програмний компонент відповідає їхнім вимогам.

Принцип 3. «Вітайте зміну вимог, навіть на пізніх стадіях розробки». Вимоги змінюватимуться під час розробки програмного забезпечення, щоб відобразити зміни в (1) середовищі, в якому програмне забезпечення буде

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		31

впроваджено, та в (2) середовищі розробки. Це широко визнано (як у спільноті гнучких процесів, так і за їх межами) і є однією з основних причин відмови від простої каскадної моделі. Еволюція вимог часто розглядається як невід'ємна проблема розробки програмного забезпечення [21]. Спільнота гнучких процесів розглядає зміни вимог як надання можливостей для розвитку програмного забезпечення, яке може підвищити конкурентоспроможність клієнта в швидкозмінному середовищі. Команди розробників, які можуть впоратися з такими змінами та створити програмне забезпечення, корисне для клієнта на момент поставки (а не на початку проекту), мають більше шансів мати задоволених клієнтів. Стверджується, що короткі цикли ітерацій та підхід «плануйте одну ітерацію за раз» забезпечують гнучкість, необхідну для врахування змін у гнучких процесах. Прихильники гнучких процесів стверджують, що дотримання цього принципу значно підвищує конкурентоспроможність компанії.

Припущення щодо зміни вимог: Вимоги завжди розвиваються через зміни технологій, потреб клієнтів, бізнес-сфер або навіть залучення нових клієнтів.

Припущення, яке тут робиться, по суті, є повторенням принципу. Зміна вимог не обов'язково вважається поганою, але вітається як можливість задовольнити потреби клієнтів навіть краще, ніж за умови негнучкого дотримання старих вимог. Якщо потреби клієнтів змінюються на пізніх етапах проекту, то для його успіху важливо забезпечити адаптацію проекту до цих змін.

Припущення щодо вартості змін : вартість змін не зростає різко з часом.

Гнучкі процеси кидають виклик широко поширеній думці, що помилки, внесені на ранніх стадіях та виявлені на пізніх етапах процесу, мають значно вищі витрати, ніж помилки, виявлені на ранніх стадіях. Прихильники гнучких процесів стверджують, що належне використання нових технологій та практик розробки може зменшити вартість виявлення помилок на пізніх етапах процесу розробки [22]. Можна навести переконливі аргументи на користь того, що використання таких технологій та практик, як (1) дуже швидкі компілятори зі складними системами типів, (2) інтегровані середовища розробки, (3)

систематичне вдосконалення коду шляхом рефакторингу та (4) автоматизовані набори тестів, може допомогти керувати витратами на виявлення та видалення помилок, навіть коли помилки виявляються на пізніх етапах процесу. Також очевидно, що вартість виправлення помилок, які можна виправити за допомогою локалізованих змін, тобто змін з обмеженим впливом, повинна бути відносно стабільною з часом. З іншого боку, також очевидно, що певні типи помилок, наприклад, недоліки архітектурного проєкту, які серйозно порушують цілісність проєкту, або помилки, які потребують коригувальних дій, що мають широкий вплив, є більш витратними для виправлення, чим пізніше вони виявляються.

Принцип 6. «Створюйте проєкти навколо мотивованих людей. Забезпечте їм необхідне середовище та підтримку, і довірте їм виконання роботи». Принцип 7. «Найкращі архітектури, вимоги та дизайн виникають у самоорганізованих командах».

Принцип 12. «Проектні команди регулярно оцінюють свою ефективність та відповідно коригують свою поведінку».

Такі гнучкі процеси, як XP та Scrum, підкреслюють необхідність захисту розробників від відволікаючих факторів, щоб вони могли зосередитися виключно на проєктній діяльності. Роль керівництва полягає у сприянні розробці, забезпечуючи розробникам необхідні ресурси, коли це необхідно, і не відволікаючи їх на проблеми поза межами проєкту. Керівництво також повинно утримуватися від нав'язування та мікроменеджменту команди розробників: розробникам слід довіряти виконання роботи за допомогою процесу, що базується на їхньому колективному досвіді (тобто команда повинна бути самоорганізованою) [23]. Мотивація є однією з найважливіших властивостей, необхідних людям для досягнення амбітних цілей з якісними результатами.

Перетворення традиційної команди на гнучку, самоорганізовану команду може бути складним. У деяких гнучких процесах це може вимагати від керівників команд передачі частини своїх традиційних обов'язків членам команди. Короткі ітерації гнучких процесів дозволяють керівнику проєкту перевірити передачу відповідальності та таким чином поступово зміцнити довіру

до здатності команди виконати роботу.

Стверджується, що підтримка Принципу 6 призводить до продуктів вищої якості, які відповідають вимогам клієнтів під час доставки, є краще структурованими та потребують менше зусиль для створення, ніж ті, що створені з використанням більш прогнозованих (важких) процесів. Однак нам невідомі жодні емпіричні дослідження, які б свідчили про покращення якості та зменшення зусиль в результаті використання гнучких процесів.

Часті огляди, що рекомендуються гнучкими процесами, зосереджені на продуктах та процесах, що використовуються для їх розробки. Планування ітерацій також дозволяє аналізувати попередні результати та коригувати майбутні ітерації. Оскільки клієнт постійно залучений, можна отримати різні точки зору на ефективність команди проекту та гнучко реагувати на ці роздуми. Гнучкість у гнучких процесах досягається шляхом самоаналізу використовуваних процесів та відповідної адаптації процесу.

Однак для самооцінки та коригування проекту потрібне середовище проекту, яке дозволяє гнучку адаптацію. Якщо середовище «вороже», це означає, що воно негнучке до змін, його клієнти не бажають активно брати участь, його підрядники наполягають на виконанні письмових специфікацій тощо. Стає набагато складніше діяти гнучко.

Припущення щодо досвіду роботи в команді : Розробники мають досвід, необхідний для належного визначення та адаптації своїх процесів.

Іншими словами, організація завжди може сформувати команду, що складається з розумних, досвідчених фахівців з вирішення проблем, здатних ефективно розвивати свої процеси. Команда розробників, яка (1) складається з розробників з ґрунтовними навичками програмування та досвідом роботи з відповідними процесами та продуктами, і (2) має здатність до конвергенції шляхом раціональних обговорень, ймовірно, зможе ефективно визначити та адаптувати свої проекти. На жаль, не всі команди розробників мають ці якості. Деяким потрібне керівництво у визначенні відповідних процесів. Для таких команд «стандартний» процес може працювати краще, ніж адаптивний процес,

який їм може бути важко контролювати [24]. Дійсно, припущення про досвід команди є критично важливим для успіху гнучких проектів розробки.

Загальновизнано, що не існує єдиного процесу, який би підходив до всіх проектів. З іншого боку, існує низка найкращих практик, методик та досвіду, які розробники можуть використовувати у відповідних ситуаціях. Команди розробників програмного забезпечення, що складаються з провідних членів, які розуміють ситуації, в яких застосовні певні процеси та практики, мають більше шансів на успіх у гнучкому середовищі. Тому відповідальність майбутніх гнучких розробників полягає в тому, щоб розвинути таке розуміння, набуваючи досвіду з різними підходами. Команди, що складаються з розробників з такими навичками, мають більше шансів отримати користь від використання гнучких процесів.

У рамках припущення про командний досвід, здається, є ще два припущення: припущення про самооцінку та припущення про самоорганізацію.

Припущення щодо самооцінки : команди здатні та бажають оцінювати себе.

Команда повинна оцінити свій процес, якщо вона сподівається адаптувати та/або покращити його. Припущення полягає в тому, що команда здатна та бажає це зробити. Це складно в проектній культурі, де неоптимальна поведінка розглядається як серйозна проблема, і тому члени команди можуть неохоче проводити чесні самооцінки. Крім того, навіть якщо команда готова до самооцінки, вона також повинна мати для цього навички. Це, по суті, зводиться до необхідності того, щоб члени команди мали досвід у попередніх успішних проектах, щоб мати змогу порівняти ефективність цього проекту з попередніми та визначити можливі покращення.

Припущення самоорганізації: Найкращі архітектури, вимоги та дизайн виникають у самоорганізованих командах.

Припущення тут полягає в тому, що самоорганізовані команди не лише створюють найкращі архітектури, методи виявлення вимог та дизайн, але й що існують ресурси для створення самоорганізованих команд, а також що

керівництво дозволяє та підтримує такий підхід.

Хоча це припущення, по суті, є переформулюванням Принципу 7, його не слід розглядати як просто зайве [25]. Передбачається, що команди будуть самоорганізуватися, залучаючи найкваліфікованіших фахівців, створюючи таким чином команди з різноманітними здібностями, а отже, і здатність створювати найкращі можливі продукти. Концепція самоорганізованих команд дуже відрізняється від того, як працює багато організацій. Таким чином, якщо організація очікує отримати максимальну користь від застосування Agile-процесів, вона повинна усвідомлювати, що її управління командами, можливо, потребуватиме радикальної переробки .

Принцип 9. «Гнучкі процеси сприяють сталому розвитку».

Принцип 10. «Постійна увага до технічної досконалості та гарного дизайну підвищує гнучкість». Використовуючи гнучкі процеси, розробники зосереджуються на наданні лише необхідної функціональності та своєчасному розвитку програмного забезпечення у відповідь на зміни в потребах клієнтів і ринку. Прихильники гнучких процесів наголошують на важливості сприяння середовищу розробки, яке постійно стимулює та мотивує розробників. Такі правила, як 40-годинний робочий тиждень у XP та «Без понаднормової роботи», спрямовані на цей принцип.

Основними заходами контролю якості в Agile-процесах є тестування коду та зворотний зв'язок з клієнтами [26]. У таких процесах, як Scrum, рекомендуються часті зустрічі для огляду, тоді як екстремальне програмування пропагує безперервні огляди шляхом парної розробки коду. Екстремальне програмування також рекомендує створення тестових випадків перед складанням коду та використання регресійних тестів, щоб переконатися, що впроваджені зміни не матимуть небажаних наслідків.

Зі зростанням систем з часом, спочатку добре розроблена архітектура може ставати дедалі розмитішою. Екстремальне програмування використовує техніку рефакторингу для постійного перероблення системи та, таким чином, підтримки оптимальної якості дизайну. Це дозволяє реалізацію вдосконалювати

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

для подальших ітерацій та підтримувати її в майбутньому.

Припущення щодо забезпечення якості: Оцінювання програмних артефактів (продуктів та процесів) може обмежуватися частими неформальними інтерв'ю, оглядами та тестуванням коду.

XP замінює традиційний огляд парним програмуванням, колективним володінням кодом та суворим підходом «спочатку тестуй». Ці підходи надають можливості для постійного огляду та вдосконалення продукту під час розробки. Scrum та Crystal рекомендують часте використання семінарів, оглядових зустрічей та співбесід для оцінки продуктів та процесу, а також використання результатів для відповідної адаптації процесу.

Незважаючи на їхні очевидні переваги, здається, що неформальні методи оцінювання гнучких процесів можуть бути недостатніми для встановлення якості критично важливих для безпеки систем – систем, в яких збій може призвести до безпосередньої травми людей або серйозної економічної шкоди. Методи розробки та тестування, які є більш формальними та/або ретельно спланованими, можуть допомогти забезпечити якість цих типів систем [27]. Однак вони вимагають значно більше зусиль і, отже, є набагато дорожчими. Наприклад, перевірка реалізації на відповідність її вимогам за допомогою методів аналізу означає, що з вимог має бути отримана точна та детальна модель специфікації.

Припущення про безперервний редизайн : Системи можна безперервно редизайнувати (рефакторувати) та зберігати їхню структурну та концептуальну цілісність. Одним з головних припущень, що лежать в основі гнучкої розробки, є те, що дизайн можна і потрібно постійно переробляти. День за днем дизайн переоцінюється, і в міру визначення кращих варіантів проводиться рефакторинг та переробка. Звичайно, важливим припущенням для цього є те, що цей редизайн можна проводити протягом значного періоду часу, не руйнуючи структурну та концептуальну цілісність дизайну та продукту.

Принцип 11. «Простота є надзвичайно важливою». Цей принцип є прямою реакцією на те, що сприймається як непотрібна складність, що

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		37

накладається важкими процесами. Тому гнучкі процеси пропагують простоту як коду, так і використовуваних інструментів. Генератори коду або фреймворки рекомендуються лише тоді, коли вони забезпечують чітку цінність для проекту. Надзвичайно важливо, щоб дизайн був простим для підтримки майбутніх ітерацій. Тому сфокусована архітектура, яка задовольняє сьогоденні потреби, є кращою за загальну архітектуру, яка «розроблена для майбутнього» [28]. Це впливає з ідеї, що майбутні зміни майже абсолютно непередбачувані, і тому немає сенсу планувати майбутнє, яке може не відбутися. Крім того, редизайн заохочується, якщо він спрощує систему та видаляє непотрібну функціональність.

В таблиці нижче підсумовано припущення, визначені в цьому розділі, які лежать в основі принципів Agile Alliance.

Припущення розробки для конкретних застосувань: можливість повторного використання та універсальність не повинні бути цілями розробки програмного забезпечення для конкретних застосувань.

Частиною простоти застосунку є зосередження на поточних вимогах і потребах, а не спроби створити більш загальну систему, яку «легше адаптувати до майбутніх потреб». Побудова більш загальної та «адаптивної» системи, як правило, робить її складнішою.

Таблиця 1.5

Припущення за принципами Agile Alliance

#	Припущення	Опис
1	Прозорість	Прозорість проекту може бути досягнута виключно через поставку працюючого коду.
2	Ітеративність	Проект завжди можна структурувати у короткі ітерації з фіксованим часом.
3	Взаємодія з клієнтом	Клієнтські команди доступні для частоті взаємодії з розробниками за потреби.
4	Комунікація в команді	Розробники знаходяться в одному часі та місці, що дозволяє їм мати часте, інтенсивне спілкування один з одним.
5	Особисте спілкування	Особиста взаємодія є найпродуктивнішим методом спілкування з клієнтами та між розробниками.
6	Документування	Розробка розлогої (відносно повної) та послідовної документації та моделей ПЗ є контрпродуктивною.
7	Змінність вимог	Вимоги завжди еволюціонують через зміни технологій, потреб клієнтів, бізнес-доменів або залучення нових клієнтів.

8	Вартість змін	Вартість змін не зростає різко з часом.
9	Досвід команди	Розробники мають необхідний досвід, щоб визначати та адаптувати свої процеси відповідним чином.
10	Самооцінка	Команди здатні та бажають оцінювати себе самі.
11	Самоорганізація	Найкращі архітектури, вимоги та дизайн виникають у самоорганізованих командах.
12	Гарантія якості	Оцінка артефактів ПЗ може бути обмежена частими неформальними інтерв'ю, ревіз'ю та тестуванням коду.
13	Специфічність розробки	Повторне використання та універсальність не повинні бути цілями розробки прикладного ПЗ.

Гнучкі процеси заохочують використання артефактів повторного використання (наприклад, фреймворків проектування, шаблонів) лише тоді, коли зрозуміло, що їх використання може допомогти зменшити витрати або підвищити якість. Створення узагальненого фрагмента коду (такого, який можна використовувати в низці ситуацій) заохочується в гнучких процесах, коли зрозуміло, що таку узагальненість можна використовувати в одному проекті (наприклад, факторизація спільних частин методів). Багато прихильників гнучких процесів стверджують, що зосередження на створенні загальних рішень може призвести до зусиль, спрямованих на те, щоб зробити системи придатними для змін, які можуть ніколи не відбутися. Це особливо стосується тих розробників, які застосовують підхід XP до гнучкої розробки; це не обов'язково є невід'ємною частиною самого принципу. Зосереджуючись на

створенні програмного забезпечення, яке реалізує конкретні вимоги, та підтримуючи цю добре продуманість, підвищується гнучкість завершення цієї розробки.

Частиною цього припущення є ідея про те, що довгострокові витрати на розробку менші, якщо в будь-який момент часу зосереджуватися на поточних вимогах, а не на узагальненні. Звичайно, це припущення є дискусійним, оскільки може виявитися, що якби початковий дизайн був більш загальним, було б легше, а отже, дешевше, додавати та адаптувати функції з часом. Але цю точку зору слід розглядати на противагу думці, що важко знати, які зміни знадобляться в майбутньому, і тому розробники можуть інвестувати в узагальнення, які ніколи не знадобляться.

Припущення про безперервний редизайн (повторний): Системи можна безперервно редизайнувати (рефакторувати) та зберігати їхню структурну та концептуальну цілісність [29].

Зазвичай, коли система вперше проектується, вона знаходиться у своєму найпростішому стані. З часом, після внесення багатьох змін, конструкція зазвичай деградує, і таким чином система стає більш «складною». Тут припускають, що це постійне перепроєктування насправді робить систему простішою.

1.4 Висновки по розділу

У першому розділі було розглянуто теоретичні основи моделювання програмних процесів у Agile-середовищі. Проаналізовано передумови виникнення та розвитку Agile-методологій, а також основні наукові підходи до моделювання процесів розробки програмного забезпечення. Досліджено ключові принципи та припущення, на яких базуються гнучкі методи управління проєктами, зокрема адаптивність, ітеративність та орієнтація на взаємодію в команді. Отримані результати формують теоретичну базу для подальшого дослідження методів і інструментів моделювання в Agile-середовищі.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ МОДЕЛЮВАННЯ В AGILE-СЕРЕДОВИЩІ

2.1 Моделювання системної динаміки процесів розробки програмного забезпечення

Моделювання все частіше використовується для розуміння процесів розробки програмного забезпечення, оскільки воно надає життєздатний лабораторний інструмент для вивчення складних явищ, які важко дослідити за допомогою інших інструментів. Моделювання використовувалося для дослідження різноманітних аспектів розробки ІС, таких як надійність

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

управління вимогами та тестування. Системна динаміка (СД) – це методологія моделювання, яка використовує принципи та методи систем керування зі зворотним зв'язком для представлення динамічної поведінки управлінських, організаційних та соціально-економічних систем. СД широко використовується в багатьох галузях досліджень, включаючи міське планування, державну політику, економіку, управління, прийняття рішень, охорону здоров'я та організаційне навчання [30].

Системна динаміка використовує засоби зворотного зв'язку для відображення взаємозв'язку між елементами в системі. Системи зворотного зв'язку можуть фіксувати взаємозв'язок між гнучкими практиками в замкнутій послідовності причин і наслідків. Модель моделювання забезпечує інтегроване середовище, в якому можна оцінити вплив гнучкої практики на весь процес розробки, імітуючи реальне середовище. Гнучкі практики тісно пов'язані одна з одною. Наприклад, рефакторинг підтримується модульним тестуванням, безперервною інтеграцією та простим проектуванням, тоді як модульне тестування підтримується парним програмуванням. Динамічна природа гнучких практик вимагає, щоб їх вивчали як інтегровану систему зворотного зв'язку, а не як окремі (ізолювані) практики. СД забезпечує середовище, в якому такий взаємозв'язок між гнучкими практиками може бути повністю представлений та вивчений.

Використання методології спрощеного розвитку (SD) у традиційному управлінні програмними проектами являє собою перше комплексне використання цього методу в дослідженнях ІС. Це дослідження розробляє розширену та інтегративну модель, яка охоплює питання управління людськими ресурсами, виробництва програмного забезпечення, а також планування та контролю. Ця модель інтегрує численні функції процесу розробки програмного забезпечення та включає як функції управлінського типу (наприклад, планування, контроль та персонал), так і діяльність типу виробництва програмного забезпечення (наприклад, проектування, кодування, огляд та тестування). Ця модель використовувалася для дослідження широкого кола

областей розробки програмного забезпечення, включаючи оцінку вартості програмного забезпечення та графік, економіку функції забезпечення якості, персонал проекту, повторне використання програмного забезпечення та управління проектами з використанням помилкової інформації. Зовсім недавно моделювання спрощеної розробки (SD) широко використовувалося в дослідженнях процесу розробки програмного забезпечення, включаючи інженерію вимог, надійне керування, аутсорсинг, інформаційну безпеку, управління знаннями та діяльність з придбання систем. Наскільки нам відомо, наша робота є першою, яка застосовує моделювання спрощеної розробки для моделювання важливих аспектів гнучкої розробки програмного забезпечення.

Для розробки структури моделі та визначення її параметрів було використано первинні дані, зібрані в рамках проектів гнучкої розробки. Для побудови та вдосконалення моделі було використано дані з дев'яти організацій. Ці організації використовують eXtreme Programming (XP) або SCRUM, або комбінацію цих двох методів. Крім того, ми також використовували вторинні джерела даних (такі як проектна документація, а також онлайн-групи новин, присвячені гнучкій розробці), які забезпечували доступ до широких обговорень гнучких практик та принципів. У таблицях 2.1 та 2.2 наведено детальну інформацію про досліджені організації та використані вторинні джерела даних. Кожне з них позначено літерним кодом. Опис нашої моделі стосується відповідних джерел даних, використаних під час її розробки.

Таблиця 2.1:

Порівняння традиційних та Agile-методів

Характеристика	Традиційні методи (Waterfall)	Agile-методи
Вимоги	Визначаються повністю на початку проекту.	Еволюціонують протягом усього проекту.
Зміни	Складно впроваджувати, потребують формальних процедур.	Вітаються та впроваджуються в будь-який час.
Зворотний зв'язок	Клієнт бачить результат лише в кінці.	Постійний зворотний зв'язок після кожної ітерації.
Комунікація	Переважно через формальну документацію.	Переважно через особисте спілкування та взаємодію.
Фокус	Дотримання плану та процесів.	Створення цінного

		працюючого продукту.
Документація	Великий обсяг детальної документації.	Мінімально необхідна документація.

Модель SD, розроблена в цьому дослідженні, інтегрує основні практики гнучкої розробки, такі як гнучке планування, короткі ітерації, залучення клієнтів, рефакторинг, модульне тестування та парне програмування. Ці практики змодельовані в чотирьох підмоделях/секторах (Рисунок 2.1): залучення клієнтів, управління змінами, гнучке планування та контроль, а також рефакторинг та якість дизайну [31]. Підсистеми управління людськими ресурсами та виробництва програмного забезпечення, необхідні для підтримки гнучкої розробки (показані сірим кольором), були створені шляхом прийняття та розширення роботи. Через обмеження простору ми надаємо короткі описи всіх підсистем і детально зупиняємося лише на рефакторингу та якості дизайну. Значне розширення системи підсистеми людських ресурсів у нашій роботі включає моделювання парного програмування, яке є унікальним для гнучкої розробки. Воно описано в онлайн-додатку.

Таблиця 2.2:

Показники успішності Agile-трансформації

Метрика	Опис	Очікуваний результат
Cycle Time	Час від початку роботи над завданням до його завершення.	Зменшення часу виходу на ринок (Time-to-Market).
Velocity	Кількість роботи, яку команда виконує за спринт.	Стабілізація та поступове зростання продуктивності.
Quality (Bug Count)	Кількість дефектів, знайдених під час розробки та після релізу.	Зменшення кількості критичних помилок.
Business Value	Цінність, яку кожна фіча приносить бізнесу.	Пріоритезація найбільш важливих функцій.
Team Happiness	Рівень задоволеності та мотивації команди.	Зменшення плинності кадрів та підвищення залученості.

Гнучка розробка спирається на пряме особисте спілкування між

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

клієнтами та розробниками для обміну знаннями. Наприклад, XP наполягає на наявності клієнта на місці, обов'язки якого включають розуміння та представлення потреб кількох сегментів клієнтів, визначення та уточнення функцій, які необхідно реалізувати, розробка приймальних тестів та перевірка правильності цих тестів, а також участь у плануванні ітерацій та релізів. Успіх проекту XP значною мірою залежить від здатності клієнта на місці належним чином виконувати ці обов'язки. Визначають кілька характеристик «хорошого» клієнта: співпрацюючий, представницький, уповноважений, відданий та обізнаний (CRACK). Гнучкі методи, такі як XP, таким чином, накладають важкий тягар на клієнта та припускають наявність таких компетентних клієнтів.

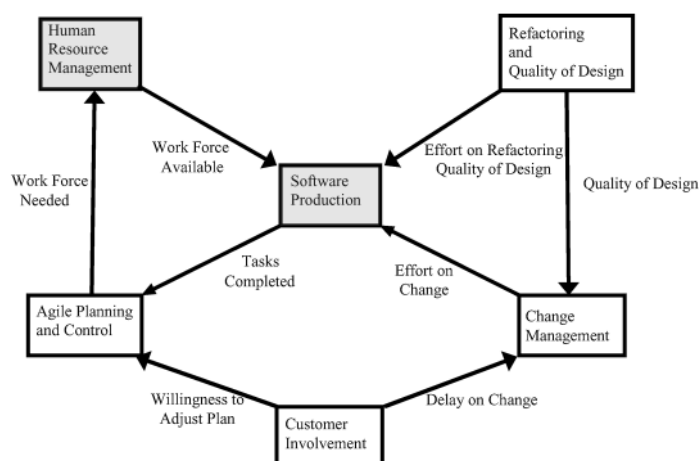
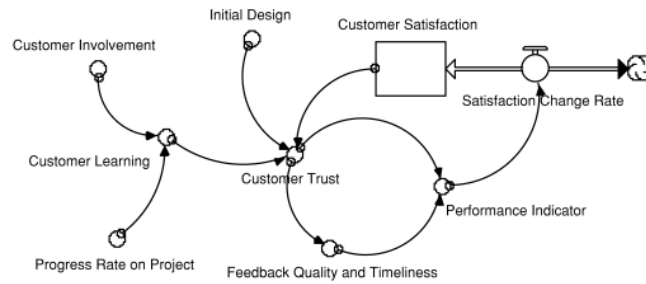


Рисунок 2.1 - Структура моделі.

Використання клієнта на місці створює кілька проблем, включаючи відсутність компетентності клієнта, ступінь залученості та конфлікт інтересів між різними клієнтами.



Legend:

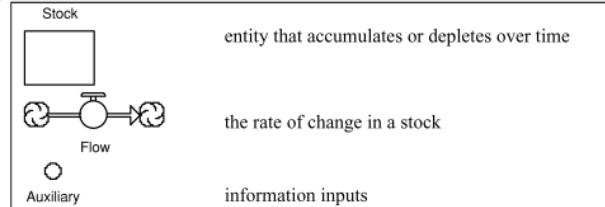


Рисунок 2.2 - Структура моделі залучення клієнтів

На рівень залучення клієнтів та швидкість виконання проекту впливає інформування клієнтів про важливі аспекти проекту та процес розробки. Навчання клієнтів, у свою чергу, призводить до довіри клієнтів. Чим вища довіра клієнтів, тим вища якість та швидкість зворотного зв'язку, що надається клієнтом команді розробників [32]. З іншого боку, затримка зворотного зв'язку може збільшити вартість проекту, оскільки зміни, внесені наприкінці життєвого циклу, важче впровадити. Збільшення вартості відображається на сприйнятій ефективності команди (змодельованій як показник ефективності). Зниження продуктивності та/або неналежно впроваджені зміни призводять до зниження задоволеності клієнтів, що, у свою чергу, впливає на довіру клієнтів. Крім того, відсутність детального початкового проектування негативно впливає на формування початкової довіри клієнтів.

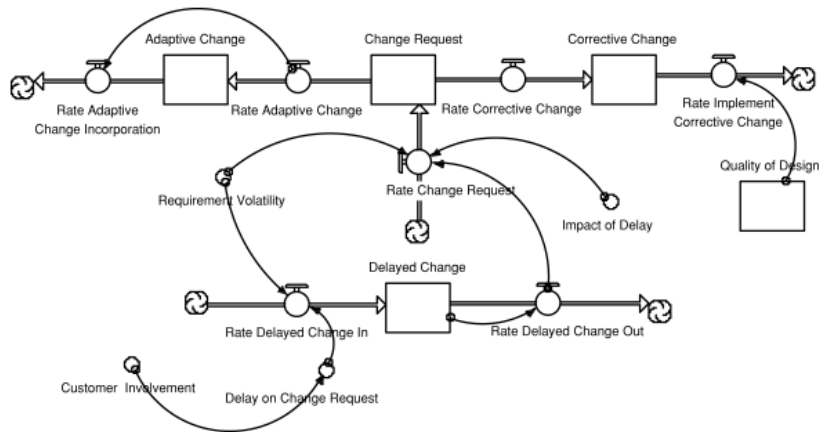


Рисунок 2.3 - Управління змінами.

2.2 Інструменти та платформи підтримки Agile-процесів і їх моделювання

Ця підсистема моделює, як гнучка розробка обробляє зміни, спричинені мінливістю вимог. Прийняття змін є основною характеристикою гнучких методів. Узгодження процесу розробки зі змінним середовищем є критичною мотивацією для впровадження гнучких методів. Оскільки до релізу додаються нові вимоги, або коли існуючі вимоги видаляються чи змінюються, це впливає на вартість, графік та якість продукту. На рисунку 2.3 показано процес управління змінами в гнучкій розробці.

Обсяг змін визначається волатильністю вимог, яка визначається як співвідношення змінених вимог (доданих, видалених та модифікованих) до загальної кількості вимог. Зміни природно поділяються на три основні класи: адаптивні, коригувальні та досконалі зміни. У гнучкій розробці досконалі зміни зазвичай впроваджуються шляхом рефакторингу. Тому в цій підсистемі моделюються лише адаптивні та коригувальні зміни. Запити на адаптивні зміни розглядаються як нові вимоги, заплановані на майбутні ітерації, якщо запитувана зміна суттєво не впливає на обсяг, графік та обмеження вартості, встановлені замовником або керівництвом [33].

Запити на коригувальні зміни впроваджуються в порядку їхнього пріоритету. Швидкість впровадження коригувальних змін залежить від якості

проекту (як описано в підсистемі: ре факторинг та якість дизайну). Клієнт переглядає функції одразу після їх впровадження та пропонує зміни, які необхідно включити в наступну ітерацію. Однак, своєчасність, з якою виявляються зміни, залежить від залученості клієнта. Більш залучений клієнт запитуватиме зміни раніше, тоді як менш залучений клієнт може усвідомити необхідність змін лише з певною затримкою. Як література, так і наші інтерв'ю підтвердили значний вплив затримки зворотного зв'язку з клієнтами (N3, N4, D1, F1, F2). Тому на будь-якому етапі проекту коригувальні зміни, запитувані клієнтом, включають зміни до функцій, які щойно були розроблені, та тих, що були розроблені в попередніх ітераціях. Затримки змін вимагають більше зусиль для впровадження через їхній хвильовий ефект, який моделюється змінною «Вплив затримки».

Підсистема планування та контролю (Рисунок 2.4) коригує графік та обсяг проекту залежно від його прогресу. Вона включає два критичні процеси: коригування обсягу та коригування графіка.

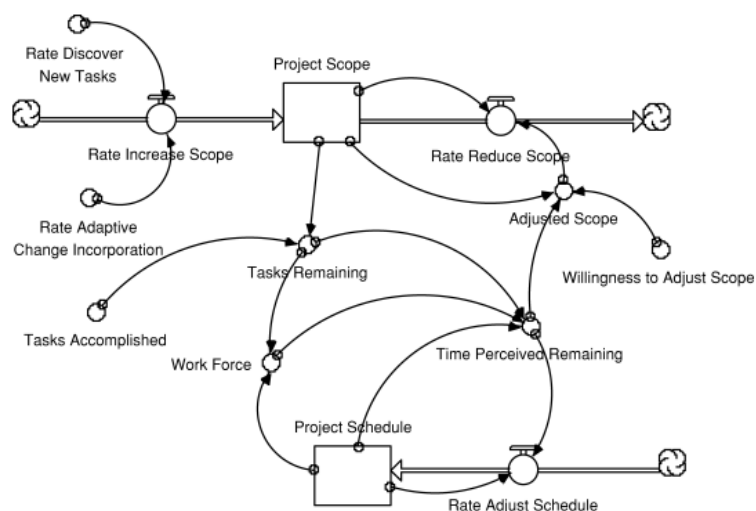


Рисунок 2.4 - Планування та контроль.

Коригування обсягу. На відміну від традиційної розробки, у гнучких проектах розробки обсяг є додатковим важелем, яким можна легко маніпулювати. Оскільки клієнти постійно надають зворотний зв'язок і запитують нові функції, обсяг проекту постійно коригується. Раніше заплановані функції

можна видалити або відкласти, коли проект відстає від графіка. У гнучкому плануванні обсяг проекту коригується наступним чином (особливо під час роботи під тиском графіка).

(1) Функціональність системи поділяється на дві групи: обов'язкові або найважливіші функції (історії), які необхідно реалізувати в першу чергу, та корисні функції, які можна реалізувати пізніше. Під тиском графіка корисні функції часто відмовляються від них.

(2) Історії розбиваються на дрібніші частини, які називаються завданнями. Виконуються лише критичні завдання, а інші відмовляються, коли проект перебуває під тиском графіка.

Коли команда усвідомлює, що обсяг проекту потрібно скоригувати для дотримання графіка виконання, вона проводить переговори із замовником, щоб визначити функції, які можна виключити. «Готовність клієнтів коригувати обсяг» залежить від «довіри клієнтів», яка описана в підсистемі залучення клієнтів.

Коригування розкладу. У гнучкій розробці програмного забезпечення розклад коригується на основі наступних обмежень. Якщо проект має фіксовану дату завершення та фіксовані ресурси, то графік фіксується так, як було заплановано спочатку. Якщо проект відстає від графіка, обсяг необхідно зменшити, видаливши історії або завдання, оскільки зазвичай важко змінити робочу силу (і опосередковано бюджет).

Подібно до традиційного проекту розробки, Agile-проект спочатку коригує графік, а потім вартість. Потім команда Agile-розробників часто має можливість узгодити зміни в обсягу проекту з клієнтом, скасувавши або відклавши деякі завдання з поточного релізу.

Ця підсистема (показана на рисунку 2.5) моделює рефакторинг та його вплив на якість дизайну. У високомінливих середовищах детальне попереднє проектування вважається марнотратним, оскільки значна частина спочатку визначеної функціональності може ніколи не бути повністю реалізована через зміни у вимогах. Рефакторинг використовується для підтримки якості дизайну

на прийнятному рівні, який відповідає поточним потребам. Ця практика, яку часто називають реструктуризацією, використовувалася для зменшення складності програмного забезпечення шляхом поступового покращення внутрішньої якості програмного забезпечення [34]. У гнучкій розробці інтенсивність, з якою практикується рефакторинг, особливо через відсутність детального початкового дизайну, робить його дуже критичним та унікальним. Визначено рефакторинг як «[зміну], внесену до внутрішньої структури програмного забезпечення, щоб зробити його легшим для розуміння та дешевшим для модифікації без зміни його спостережуваної поведінки». - Наприклад, в об'єктно-орієнтованій розробці ключовою ідеєю рефакторингу є перерозподіл класів, змінних та методів по всій ієрархії класів, щоб полегшити майбутні адаптації та розширення. Використовуючи рефакторинг, розробник програмного забезпечення може покращити дизайн програмного забезпечення, зробити його легшим для розуміння, знаходити помилки та швидше розробляти програми. Гнучкі методи покладаються на рефакторинг для безперервного вдосконалення дизайну під час розробки, замість того, щоб витрачати багато зусиль на попереднє проектування. Стверджено, що рефакторинг «призводить до програми з дизайном, який залишається хорошим під час розробки». Безперервний та інкрементальний рефакторинг – це механізм, що використовується для внесення частих змін.

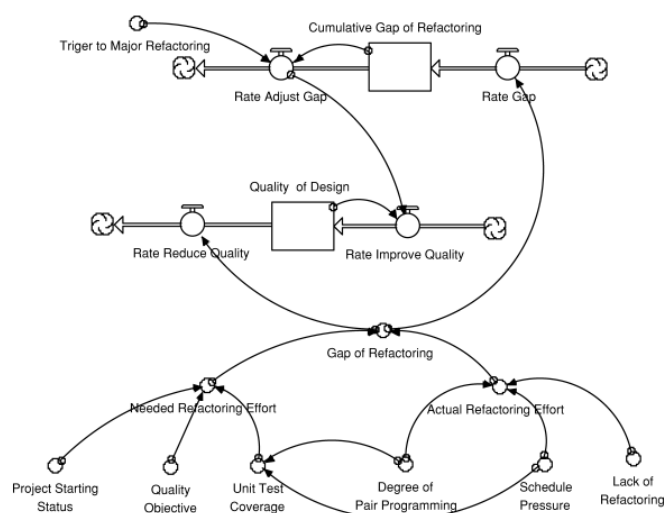


Рисунок 2.5 - Рефакторинг та якість проектування.

Змн.	Арк.	№ докум.	Підпис	Дата

БР.ІІ – 50.00.00.000 ПЗ

Арк.

49

Необхідні зусилля на рефакторинг. Рефакторинг споживає значну частину зусиль розробки, якщо його проводити регулярно. Емпіричні дослідження рефакторингу оцінюють, що фактичні зусилля коливаються від 6% до 39% від загальних зусиль розробки. Також ці дослідження зазначають, що багато програмістів поєднують рефакторинг та нову роботу, тому важко виміряти фактичний обсяг зусиль на рефакторинг. Наші дані (NA1, NA2, DC) також свідчать про те, що розробникам важко відокремити рефакторинг від іншої розробки. Перед впровадженням нової функції розробники часто рефакторують існуючий код. У моделі ми використовуємо середні зусилля, про які повідомляли учасники дослідження, які на рівні 30% представляють значну частину загальних зусиль розробки.

Три фактори впливають на зусилля, необхідні для рефакторингу: цілі якості, початковий стан проекту та охоплення модульних тестів. Цілі якості стосуються очікуваних цілей якості проекту. Початковий стан стосується якості початкового дизайну до початку кодування [35]. Модульні тести забезпечують захист від регресійних тестів та валідаційних тестів, щоб рефакторинг та інтеграція могли бути виконані ефективно. Без достатнього охоплення модульних тестів потрібно більше зусиль для валідації коду, який може змінитися через рефакторинг. Модульні тести використовуються для перевірки коректності певного модуля вихідного коду. Поширеною практикою є написання тестових випадків для кожної нетривіальної функції або методу в модулі. Модульні тести пишуться до або разом з виробничим кодом; вони підтримуються разом з виробничим кодом і використовуються кожним розробником, який працює над програмним забезпеченням.

Зусилля, необхідні для внесення однієї зміни, набагато вищі, якщо не проводиться модульне тестування. Один розробник (C1) описав свій досвід проекту без модульного тестування наступним чином: «... будь-яку зміну в одній області нам доводилося тестувати три або чотири варіанти використання, оскільки ми повторно використовували код якомога частіше. Тож ви робите одну зміну, вам доводиться пройтися через три або чотири варіанти використання. І

це уповільнювало нас, тому що у нас не було модульних тестів». Зв'язок між покриттям тестами та впевненістю розробника в програмному забезпеченні, а також зусиллями, необхідними для внесення змін, не є лінійним. Один розробник (N2) описує свій досвід проекту, який частково охоплював модульне тестування, наступним чином: «перевага від модульного тестування не пропорційна покриттю модульних тестів. Якщо покриття модульних тестів низьке, то переваги значно низькі». Фактичне покриття модульних тестів залежить від запланованого покриття, а також від ступеня тиску графіка та парного програмування. Під тиском графіка розробники скорочують модульне тестування та більше зосереджуються на кодуванні. Розробники, що працюють у парах, зазвичай більш дисципліновані, ніж окремі розробники, у написанні модульних тестів.

Фактичні зусилля на рефакторинг. Розробники майже ніколи не витрачають достатньо зусиль на рефакторинг. У моделі недостатній рефакторинг моделюється за допомогою змінної «брак рефакторингу». Наші інтерв'ю (S3, S5, H1, B5) з Agile-розробниками підтвердили цю тенденцію. Наші дані (S3, S5, H1, B5, PD, UM) також підкреслили вплив тиску розкладу та парного програмування на рефакторинг. Вплив тиску розкладу на зусилля на рефакторинг спостерігався в кількох дослідженнях. У наших інтерв'ю (N4, E1) керівники проектів визнали, що рефакторинг буде значною мірою ігноруватися під час роботи за жорстким графіком. Також попередні дослідження зазначають, що парний тиск може змусити розробників часто рефакторувати код. Наші дані (S1, S2, S6, H2) свідчать про те, що розробники зменшують свою активність рефакторингу приблизно на 50%, якщо не проводити його в парі з іншими. Коли проект наближається до дедлайну, розробники вирішують не проводити рефакторинг, оскільки підвищення продуктивності від рефакторингу буде реалізовано лише після дедлайну.

Розрив між необхідними та фактичними зусиллями з рефакторингу, який можна вважати «технічним боргом», призводить до нечистого коду, що, у свою чергу, призводить до пошкоджених проектів. Оскільки вимоги змінюються під

час розробки, проект не оновлюється, щоб врахувати нові вимоги, оскільки зміна проекту передбачала б рефакторинг великих обсягів коду. Тому зміни вносяться максимально доцільним чином, і проект системи стає дедалі пошкодженішим та крихкішим [36]. Потім розробникам доводиться виконувати «значний рефакторинг», тобто інвестувати багато зусиль для очищення коду. Відкладений рефакторинг вимагає додаткових зусиль для очищення коду пізніше через велику кількість задіяних функцій. Якість дизайну. У гнучкій розробці стверджується, що «вихідний код – це дизайн», діяльність з дизайну є частиною процесу програмування, і дизайн розвивається разом з кодом. Вплив рефакторингу на зручність підтримки оцінювався за допомогою таких метрик, як зв'язування. Однак у гнучкій розробці Кокберн зазначає, що «немає узгодженого показника якості дизайну. Тому немає однозначного виміру, за яким можна «покращувати». Якість дизайну залежить (не лише від упереджень глядача) від обставин, у яких він знаходиться – його призначення та майбутнього життя». З цієї причини в моделі ми представляємо сприйняття якості дизайну як змінну зі значеннями від 0 до 1. Якість дизайну моделюється як запас [37]. Початкове значення встановлено на 1, що означає, що на початку проекту якість дизайну знаходиться на бажаному рівні, оскільки не було проведено жодних дій, які погіршують дизайн. У міру просування проекту цей високий рівень якості необхідно підтримувати за допомогою рефакторингу. Недостатній рефакторинг знижує якість дизайну, тоді як значний рефакторинг повертає якість дизайну до прийняттого рівня.

2.3 Моделювання процесів командної взаємодії та управління ресурсами в Agile

Підсистеми управління людськими ресурсами та виробництва програмного забезпечення. Значним розширенням є моделювання парного програмування, яке є унікальним для гнучкої розробки. Досліджено роль співпраці в дослідженні 15 штатних досвідчених програмістів, які працювали над

складною проблемою. Це дослідження робить висновок, що співпраця покращує як продуктивність програмістів, так і їхнє задоволення від процесу вирішення проблем. Проведено структурований експеримент, порівнюючи парне програмування з сольним програмуванням в умовах класу. Результати показують, що парне програмування дає кращий продукт з вищою якістю за менший час, і що розробники, які використовують цей підхід, більше насолоджуються процесом. Однак, парне програмування виділяє двох осіб на кожне завдання, що збільшує витрати на оплату праці. Тому його економічна ефективність протягом життєвого циклу проекту потребує вивчення.

— *Парне програмування та продуктивність.* Хоча парне програмування використовує двох людей у кожному завданні, деякі дослідження показують, що парні розробники витрачають менше часу на кожне завдання. Результати емпіричних досліджень загального впливу парного програмування на продуктивність неоднозначні. Тому ми використовували дані наших інтерв'ю (N1, N3, E2, C1), які свідчать про те, що парне програмування в типовому проекті вимагає приблизно на 30% більше зусиль, ніж сольне програмування. — *Парне програмування та переробка.* Попередні дослідження показують, що парне програмування призводить до зменшення кількості дефектів на 40-90%. Наші дані (S1, S2, S4, E1), зібрані з реальних проектів, свідчать про те, що парне програмування призводить до зменшення кількості дефектів більш ніж на 50%. — *Парне програмування та рефакторинг і модульне тестування.* Як обговорювалося раніше, парне програмування також впливає на зусилля, витрачені на рефакторинг та модульне тестування. Наші інтерв'ю (S1, N3) показують, що коли розробники працюють самостійно, ймовірність того, що вони будуть займатися рефакторингом, зменшується вдвічі тому що вони навряд чи будуть такими ж дисциплінованими, як парна команда. Аналогічно, Парні розробники більш дисципліновані, ніж окремі розробники, у написанні модульних тестів [38]. Наші дані (N4, S3) свідчать про те, що коли розробники працюють індивідуально, вони зменшують охоплення модульного тестування.

Підсумовуючи, наша модель SD відображає основні практики, що використовуються в гнучкій розробці. Вона представляє гнучку розробку як інтегровану систему практик, а не ізольовано, та включає динамічні функції, такі як зворотний зв'язок, часові затримки та нелінійні причинно-наслідкові зв'язки. Таким чином, цю модель можна використовувати для дослідження динамічної природи впливу різних практик на критичні результати проекту, такі як вартість, графік та якість.

Наша модель пройшла ретельну валідацію після її розробки та вдосконалення. Валідація моделі спрямована на створення достатньої впевненості в моделі, щоб її результати були прийнятними. Дослідження системної динаміки зосереджуються на широкому спектрі тестів, включаючи тести структури моделі та здатності моделі відтворювати реальну поведінку. Визначення відповідної структури є першим кроком у встановленні валідності моделі системної динаміки, оскільки структура визначає її поведінку. Після того, як структурна валідність достатньо встановлена, оцінюється валідність поведінки, тобто наскільки добре поведінка, згенерована моделлю, імітує спостережувану поведінку реальної системи, для досягнення загальної валідності моделі. В онлайн-додатку ми детально повідомляємо про процедури та результати нашої розширеної валідації моделі. Спочатку представлені результати структурної валідації. У своїй фундаментальній роботі визначає десяток тестів, які можуть бути використані для структурної валідації. Різноманітні тести, що зосереджені на різних аспектах моделі, використовуються для того, щоб переконатися, що модель відповідає меті, для якої вона була розроблена. Слід зазначити, що не існує єдиного тесту, який можна було б використовувати для валідації або спростування структури моделі. У нашій валідації було використано наступні дев'ять тестів, що були релевантними для дослідження: адекватність меж, оцінка структури, розмірна узгодженість, оцінка параметрів, екстремальні умови, аномалія поведінки та несподівана поведінка. Результати всіх тестів структурної валідації свідчать про те, що структура нашої моделі точно відображає реальні процеси гнучкої розробки. Потім ми провели

поведінкову валідацію, використовуючи дані з незалежного тематичного дослідження реального проекту гнучкої розробки. Метою тематичного дослідження було вивчення здатності моделі відтворювати динамічні закономірності, що спостерігаються в проекті гнучкої розробки програмного забезпечення. Поведінкова валідація була проведена з використанням даних від RealSoft, розташованої в штаті Скелястих гір [39]. Ця компанія не лише є основним постачальником інструментів, що використовуються в гнучкій розробці, але й використовує ці інструменти та гнучкі методи у своєму процесі розробки. Проект дотримувався ітеративного підходу, кожна ітерація тривала два тижні. На момент збору даних проект триває та має плани щодо ще кількох релізів у майбутньому. Команди розробників склалися з чотирьох розробників, одного інженера з контролю якості та одного менеджера продукту (який також виступав у ролі заступника клієнта). Інші відповідні деталі щодо організації та проекту описані в онлайн-додатку.

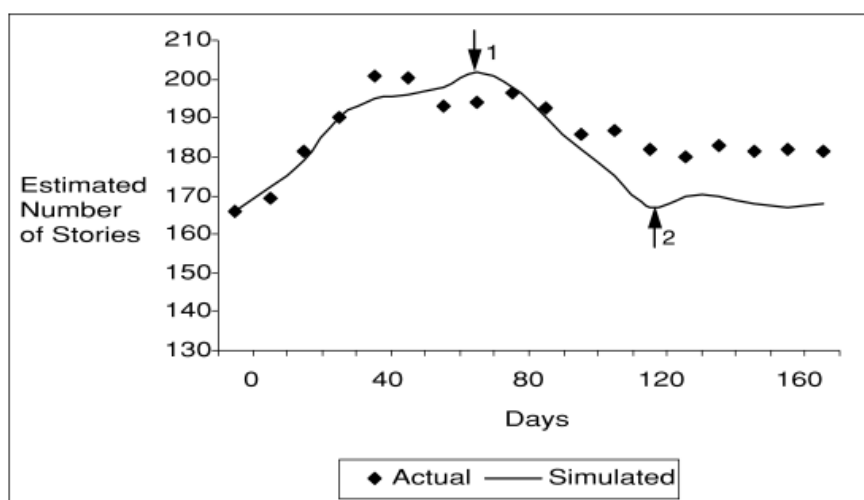


Рисунок 2.6 - Орієнтовна кількість поверхів (розрахована та фактична) у проекті RealSoft.

Параметри моделі були встановлені на значення, що відображали середовище RealSoft. Після параметризації моделі її було запущено для моделювання проекту RealSoft у середовищі моделювання STELLA®. Ми дослідили динамічну поведінку наступних трьох критичних змінних проекту: (1)

розрахункова кількість поверхів, (2) продуктивність та (3) кількість дефектів. Ми порівнюємо змодельовану поведінку моделі з часом з фактичними результатами проекту. Модель точно відтворює фактичну поведінку проекту RealSoft у трьох досліджуваних змінних. Через обмеження обсягу ми представляємо результати дослідження лише першої змінної. Решта результатів представлені в онлайн-додатку.

Орієнтовна кількість поверхів. На рисунку 6 зображено фактичну та змодельовану розрахункову кількість поверхів. На рисунку видно, що розрахунковий обсяг проекту змінювався з часом. Зростання та зменшення обсягу зумовлені двома факторами, які впливають на коригування обсягу в протилежних напрямках: (1) розширення обсягу: обсяг має тенденцію до збільшення, оскільки з'являються нові завдання та клієнт запитує нові функції, (2) скорочення обсягу: обсяг зменшується, щоб вкласти в термін, оскільки - проект вважається таким, що відстає від графіка. Обсяг розширюється або зменшується залежно від фактора, який має сильніший вплив. У цьому випадку розширення обсягу є домінуючим протягом перших 80 днів (позначено стрілкою 1), оскільки дата завершення продовжується протягом цього періоду. Однак, коли досягнуто максимально допустимої дати завершення, вплив обмеження графіка є більшим, і обсяг зменшується, щоб вкласти в заплановану дату здачі.

Модель оцінює кількість історій у 168, що трохи менше, ніж фактична кількість для двох останніх релізів (181). Ймовірною причиною цієї незначної різниці є те, що модель припускає, що проект завершується в кінці змодельованих циклів. Ближче до кінця проекту керівництво з більшою ймовірністю коригуватиме обсяг проекту, щоб врахувати терміни. У моделі готовність клієнта змінювати обсяг проекту є змінною, яка зростає з часом. Однак RealSoft був поточним проектом, і більше релізів перебували в плануванні або розробці. Як змодельований обсяг, так і фактичний обсяг знову піднімаються вгору приблизно на 120-й день (позначено стрілкою 2). Ретельне вивчення поведінки моделі показує, що це пов'язано з динамікою продуктивності [40]. Після значного рефакторингу, проведеного до 100-го дня проекту, якість дизайну

значно покращується, а продуктивність зростає протягом чотирьох ітерацій. Підвищена продуктивність призводить до збільшення розрахункової кількості завдань, які можна виконати.

Підсумовуючи, поведінкова валідація демонструє, що модель моделювання SD, розроблена в нашому дослідженні, чітко прогнозує динамічну поведінку критичних результатів проекту. У поєднанні з тестами для валідації структури, ця поведінкова валідація встановлює, що модель є валідною та дуже добре відображає типовий проект гнучкої розробки програмного забезпечення. Хоча жоден з цих тестів окремо не є достатнім для забезпечення валідності моделі, у поєднанні вони забезпечують потужний фільтр для оцінки продуктивності моделі. Результати добре зарекомендували себе тестів, проведених у нашому дослідженні, підтверджують, що наша модель є достатньо адекватною та придатною для використання для дослідження динаміки гнучкої розробки програмного забезпечення.

2.4 Висновки по розділу

У другому розділі було досліджено методи та інструменти моделювання програмних процесів у Agile-середовищі. Розглянуто підходи до моделювання системної динаміки процесів розробки програмного забезпечення, що дозволяють аналізувати та прогнозувати поведінку проєктів. Проведено огляд сучасних інструментів і платформ, які підтримують Agile-процеси та їх моделювання. Також досліджено моделювання процесів командної взаємодії та управління ресурсами, що є важливими складовими ефективної організації Agile-розробки. У результаті визначено практичні методи підвищення керованості та ефективності програмних процесів.

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТА ОЦІНКА ЕФЕКТИВНОСТІ AGILE-ПРОЦЕСІВ

3.1 Результати моделювання та експериментальних досліджень

Після валідації моделі ми провели експеримент для вивчення двох критично важливих практик у гнучкій розробці програмного забезпечення. Зокрема, ми дослідили роль та вплив рефакторингу на продуктивність проекту та економіку парного програмування, використовуючи дані з проекту RealSoft. Проект RealSoft був обраний, оскільки він представляв собою типовий проект гнучкої розробки середнього розміру, що дотримується гібридної методології, що використовує практики XP та SCRUM. Параметри моделі були встановлені на значення, що відображають середовище проекту RealSoft. Модель гнучкої розробки була визначена в середовищі STELLA®, і її поведінка була змодельована для розуміння динамічних ефектів рефакторингу та парного програмування. Цей експеримент ілюструє використання нашої моделі симуляції гнучкої розробки як «симулятора польоту» для проектів гнучкої розробки. Керівники проектів ІС та персонал проектів можуть використовувати нашу симуляцію гнучкої розробки для вивчення впливу різноманітних практик гнучкої розробки та політик управління, як показано в наступних підрозділах.

У реальних проектах необхідність рефакторингу не завжди очевидна. Крім того, ручне виконання цього завдання займає досить багато часу. Крім того, під тиском графіка програмісти не будуть виконувати рефакторинг. Так само досвідчені програмісти ненавидять модифікувати код, який працює, боячись створити неочікувані помилки. Афоризм «якщо воно не зламане, не лагодь» є нормою. Розробники можуть надмірно зосереджуватися на забезпеченні функціональності та нехтувати критичними потребами рефакторингу. Як результат, фактичні зусилля на рефакторинг майже завжди менші, ніж необхідні зусилля на рефакторинг.

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		58

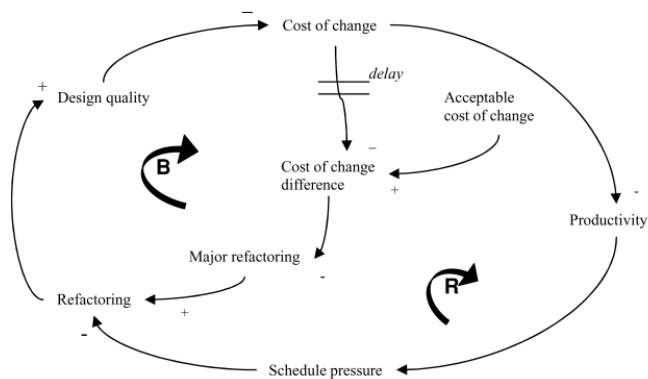


Рисунок 3.1 - Структури зворотного зв'язку між рефакторингом та якістю проектування

На рисунку 3.1 показано структури зворотного зв'язку рефакторингу та якості дизайну з циклом балансування (Цикл В) та циклом підсилення (Цикл R). В основі лежить цикл балансування зворотного зв'язку із затримкою. Починаючи зліва, недостатній рефакторинг призводить до низької якості дизайну. Хоча вимоги до проекту змінюються під час розробки, дизайн не оновлюється негайно, щоб врахувати нові вимоги. Зміни вносяться максимально доцільним способом, і дизайн системи стає все більш неефективним та крихким. Зі зниженням якості дизайну погіршується і здатність розуміти та ефективно розвивати код. Це не тільки ускладнює зміну програмного забезпечення, але й полегшує породження помилок, а також ускладнює їх виявлення та безпечне видалення. Погано розроблений код зазвичай вимагає більше зусиль для включення нової функції та уповільнює розробку. Тому вартість внесення модифікацій зростає. Однак існує поріг, після якого вартість модифікацій не може зростати. Розробка може тривати, доки ця вартість не стане нижчою за певний поріг. Таким чином, існує затримка усвідомлення збільшення вартості внесення модифікацій, спричиненої низькою якістю дизайну, доки не досягне точки, коли вартість перевищить поріг, або затримка у внесенні модифікацій стане неприйнятно довгою. Тоді розробникам доводиться зупиняти розробку та очищати безладний код (тобто виконувати «значний рефакторинг»). Після значного рефакторингу дизайн має прийнятний рівень якості, настільки, що розробники можуть продовжити інший цикл розробки (завершення/закриття

циклу балансування V).

Чим більше дизайн страждає від недостатнього рефакторингу, тим швидше погіршується його якість. Як наслідок, продуктивність знижується, а тиск на графік зростає. У відповідь розробники витрачають менше часу на рефакторинг і зосереджуються на виконанні поставлених перед ними завдань розробки, що, у свою чергу призводить до подальшого погіршення якості проекту (завершення армуючої петлі R).

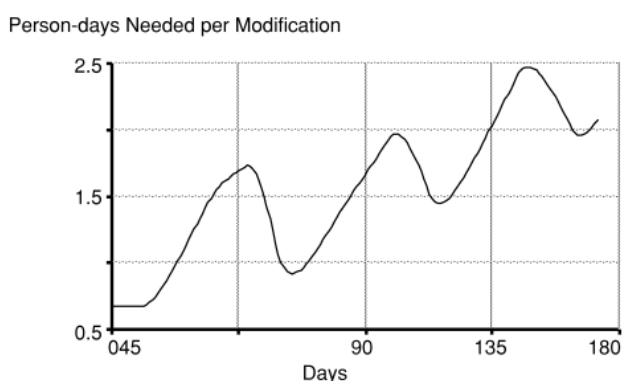


Рисунок 3.2 - Вартість внесення змін (людино-днів/змiна) у проєкті.

Далі ми обговоримо вплив рефакторингу на вартість внесення модифікацій та використання парного програмування. Давно відомо, що в традиційній розробці вартість модифікації програмного забезпечення зростає експоненціально з часом. Тому традиційні методи підкреслюють важливість точних вимог та детального попереднього проектування. У гнучкій розробці клієнт надає постійний зворотний зв'язок, а функції, які необхідно модифікувати, визначаються одразу після їх розробки. Отримані зміни, ймовірно, будуть невеликими та недорогими для впровадження, і в результаті стверджується, що вартість внесення модифікацій є постійною, а не зростає експоненціально. Однак, оскільки це твердження не було належним чином підтверджено попередніми дослідженнями, ми розглянемо його за допомогою нашого SD-модельовання проекту RealSoft.

Крива вартості змін, отримана за моделлю, показана на рисунку 8. Вона показує людино-дні, необхідні для кожної зміни. Вартість кожної зміни зростає

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		60

з часом, і що цікавіше, криві піднімаються та опускаються протягом кількох циклів. Крім того, ми виявляємо, що вартість змін швидко зростає протягом кожного невеликого циклу. Це збільшення безпосередньо пов'язане з необхідністю рефакторингу. В ідеалі розробники повинні продовжувати рефакторинг коду, щоб підтримувати якість дизайну. Однак багато факторів, таких як тиск графіка, ступінь парного програмування та, найголовніше, відсутність мотивації, впливають на фактичний ступінь рефакторингу. Недостатній рефакторинг знизить якість дизайну, і тому з часом виконання запитів на зміни стає складним, доки розробники не почнуть очищати безладний код, проводячи значний рефакторинг. Дизайн значно покращується завдяки значному рефакторингу, і тоді вартість кожної зміни зменшується. Однак потреба в ще одному циклі рефакторингу починається одразу після основного рефакторингу. На рисунку 3.3 показано, як якість дизайну змінюється з часом за умови недостатнього рефакторингу.

Вартість однієї зміни (кількість людино-днів, необхідних для кожної зміни) коливається від 0,65 до 2,47 людино-днів. Наш експеримент показує, що в типовому Agile-проекті:

(1) вартість змін не є постійною, а радше її значення змінюється циклічно;

(2) Загалом, вартість змін зростає з часом, але значно меншою мірою (4:1), ніж у традиційному підході (100:1). Результат має значні наслідки для гнучких методів: менший діапазон вартості змін, здається, робить ітеративну розробку можливою. Однак її застосовність значною мірою залежить від практики рефакторингу. Навіть якщо короткі ітерації та часті релізи різко зменшують масштаб змін, вартість змін може швидко зрости, якщо дизайн не вдосконалювати регулярно. Це виявляє потенційний ризик у гнучкій розробці: якщо рефакторинг не виконується ретельно, процес розробки буде неефективним.

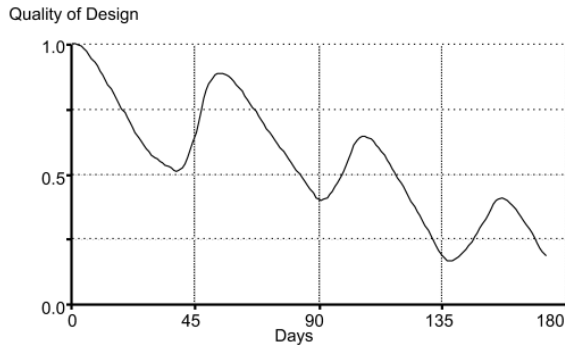


Рисунок 3.3 - Якість проектування в проекті.

Далі ми тестуємо вплив поведінки рефакторингу на вартість розробки. Масштабний рефакторинг запускається, коли якість дизайну падає до такого рівня, що зусилля, необхідні для внесення змін (наприклад, додавання нової функції або модифікації розроблених завдань), набагато перевищують звичайні зусилля. Розробники визнають цю проблему, а потім починають очищати систему, що розпочинає масштабний рефакторинг. На рисунку 3.4 показано, як затримка в рефакторингу впливає на продуктивність проекту. Затримка представлена значенням якості дизайну, коли запускається масштабний рефакторинг. Якість дизайну моделюється як змінна зі значеннями від нуля (що представляє ситуації, коли масштабний рефакторинг взагалі не проводиться) до одиниці (що представляє ситуації, коли код постійно рефакторується). Рисунок 3.4 чітко показують, що продуктивність проекту дуже чутлива до затримки в масштабному рефакторингу. У проекті з фіксованим графіком більша затримка призводить до вищих витрат і меншої кількості наданих історій. Цікаво виявити, що коли затримка перевищує певний поріг (у цьому випадку якість дизайну = 0,4), збільшення затримки призводить до зниження витрат та більшої кількості реалізованих історій. Причиною може бути те, що додаткові зусилля, витрачені на значний рефакторинг, не є економічно ефективними, якщо його виконувати після значної затримки. Моделювання показує, що загалом, чим вищий рівень рефакторингу, тим нижча загальна вартість розробки. Однак, оскільки проекти майже завжди страждають від недостатнього рефакторингу, серйозного рефакторингу не уникнути. У таких ситуаціях команди розробників повинні

планувати серйозний рефакторинг у плані випуску. На рисунках 3.3 та 3.4 показано, що серйозний рефакторинг слід планувати часто. Після того, як якість дизайну падає нижче мінімально прийнятного рівня, вартість додавання нових функцій різко зростає.

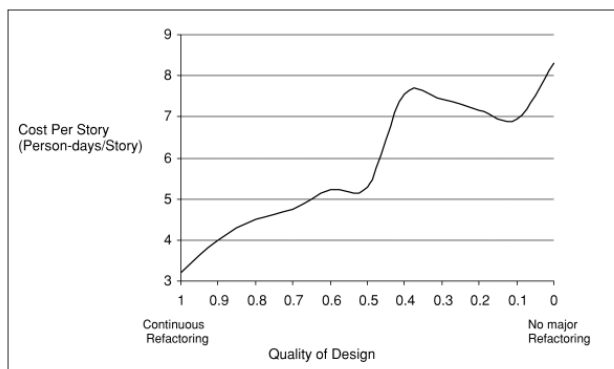


Рисунок 3.4 - Вплив затримки масштабного рефакторингу на вартість однієї історії в проєкті.

Підсумовуючи, рефакторинг є критично важливим для гнучкої розробки програмного забезпечення, а продуктивність проєкту дуже чутлива до затримок у значному рефакторингу.

3.2 Аналіз обмежень Agile-процесів та шляхи їх подолання

З обговорення в попередньому розділі має бути зрозуміло, що припущення, що лежать в основі гнучких процесів, не справедливі для всіх проєктів та середовищ розробки програмного забезпечення. Це не повинно дивувати: гнучкі підходи – це не чарівні кулі. Оскільки ці припущення виконуються не в усіх організаціях та/або середовищах розробки, гнучкі підходи в їхніх поточних формах мають обмеження. Можливо розширити гнучкі процеси для усунення їхніх обмежень. Такі розширення можуть включати включення принципів та практик, які часто асоціюються з більш прогнозними, плановими або «традиційними» процесами розробки, у гнучкі процеси. Загалом, користувачі гнучких процесів повинні забезпечити, щоб практики, засновані на

припущеннях, які не є дійсними в їхніх середовищах розробки, були відповідно змінені.

У цьому підрозділі ми визначаємо деякі обмеження, пов'язані з припущеннями, зробленими гнучкими процесами, та обговорюємо, як деякі з цих обмежень можна вирішити. Для кожного обмеження ми характеризуємо ситуації, в яких припущення, що призводять до обмеження, не виконуються, та обговорюємо, як гнучкі процеси можна модифікувати, щоб розширити застосовність гнучких процесів. Не всі припущення, визначені в попередньому розділі, безпосередньо призводять до обмежень, обговорених у цьому розділі.

В таблиці 3.1 нижче підсумовано взаємозв'язки між обмеженнями, обговореними в цьому розділі, та відповідними припущеннями, визначеними в попередньому розділі. Ми визначили дві категорії обмежень: обмеження, пов'язані з персоналом, та обмеження, пов'язані з продуктом. Припущення, орієнтовані на людей, як правило, призводять до обмежень у категорії персоналу, тоді як припущення щодо типів артефактів, що виробляються в проєкті, призводять до обмежень у категорії продукту.

Таблиця 3.1:

Порівняння ролей у Scrum

Роль	Основні обов'язки	Фокус
Власник продукту (Product Owner)	Визначення бачення продукту, управління беклогом, пріоритезація завдань.	Бізнес-цінність та потреби клієнта.
Scrum-майстер	Фасилітація процесів, усунення перешкод, допомога команді у дотриманні принципів Scrum.	Ефективність процесу та здоров'я команди.
Команда розробки	Виконання технічних завдань, створення інкременту продукту, самоорганізація.	Технічна реалізація та якість коду.

Розподілені середовища розробки – це середовища, в яких розробники не знаходяться в одному географічному місці або не знаходяться в безпосередній географічній близькості один до одного. Аналогічно, якщо команда розробників не знаходиться в безпосередній географічній близькості до клієнта, можуть

виникнути подібні проблеми.

Географічна розсіяність призводить до різних проблем, яких немає, коли всі знаходяться в одному місці або, принаймні, розташовані відносно близько один до одного (наприклад, в одному місті або в двох містах, які розташовані недалеко одне від одного). Розподілена розробка зазвичай ускладнює комунікацію, оскільки люди не можуть взаємодіяти одночасно та/або в одному місці. Навіть якщо комунікація не є складнішою, розподілена розробка вимагає спеціальних допоміжних інструментів,

технології та механізми комунікації для врахування унікальних вимог та характеристик такого середовища.

У розподілених середовищах розробки припущення щодо взаємодії з клієнтами, спілкування в команді, особистого спілкування та документації можуть бути невиконавними. Перші три припущення передбачають, що розробникам дуже легко взаємодіяти один з одним та з клієнтами. Фактично, припущення щодо особистого спілкування передбачає, що розробники та клієнти знаходяться разом, де вони можуть зустрітися віч-на-віч – що вони розташовані поруч – оскільки Agile-розробники вважають, що це найпродуктивніший спосіб взаємодії.

Географічний розподіл ускладнює взаємодію через різні графіки роботи, різницю в часових поясах, а також через те, що розробники та клієнти не завжди можуть бачити реакції один одного та обмінюватися ідеями так само гнучко та чітко. Акцент на спільному розміщенні в гнучких процесах не дуже добре відповідає прагненню деяких галузей реалізувати глобально розподілені середовища розробки програмного забезпечення. Різні витрати на робочу силу в інших регіонах або інших країнах можуть мотивувати клієнтів наймати офшорних розробників або можуть мотивувати розробників використовувати офшорну робочу силу. Середовища розробки, в яких члени команди та клієнти фізично розподілені, можуть бути не в змозі забезпечити особисте спілкування, яке пропагують гнучкі процеси. У таких випадках можна принаймні наблизитися до особистого спілкування за допомогою таких технологій, як відеоконференції,

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

чат та онлайн-дошки, конференц-дзвінки тощо, але ці технології можуть бути дорогими та не завжди такими ефективними, як можна було б сподіватися.

Особисте спілкування може бути таким же важливим у розподілених середовищах, як і в нерозподілених. Такі зустрічі необхідно планувати заздалегідь, щоб забезпечити участь усіх учасників, а обговорення були ефективними та не надто трудомісткими. Такі особисті зустрічі можна використовувати як основні заходи синхронізації, на яких географічно розподілені розробники (1) інформуються про прогрес, досягнутий іншими, та (2) обговорюють плани подальшого розвитку продукту. Між такими зустрічами документація (окрім коду) може стати основною формою комунікації, яку доповнюють технології електронної пошти, чату та відеоконференцій.

Своєчасне створення та підтримка належної документації вимог та проєктів є важливою для забезпечення однакового бачення продукту, який буде створено, у всіх розподілених членів команди. Це не слід тлумачити як вимогу документувати або моделювати всі аспекти програмного забезпечення. Документацію та моделі слід створювати та підтримувати лише за умови, що вони надають цінність проєкту та зацікавленим сторонам проєкту.

Гнучкість не завжди можлива, якщо комунікація обмежується обміном офіційною документацією з юридичних причин або через глобальну розподілену розробку. У цих випадках лише елементи гнучкого процесу можуть бути впроваджені локально, тоді як формальні процеси використовуються для координації більшого, розподіленого проєкту.

На жаль, особливо в розподілених середовищах документація є ще важливішою через різний час та місце виконання робочих завдань, а також через те, що різні люди та команди одночасно та послідовно працюють над одним і тим самим проєктом. Документація стає ще важливішою через обмежені способи взаємодії розробників та клієнтів. Таким чином, зосередженість на мінімізації документації створює обмеження в тому, наскільки добре розподілена розробка може виконуватися відповідно до гнучких процесів.

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

Аутсорсинг завдань розробки програмного забезпечення субпідрядникам часто базується на контрактах, які точно визначають, що вимагається від субпідрядника. Завдання, передані субпідрядникам, повинні бути чітко визначені у випадках, коли субпідрядники повинні подавати заявки на контракт. Під час підготовки заявки субпідрядник зазвичай розробляє план, який включає процес – з етапами та результатами – достатньо детально, щоб визначити кошторис. Процес може слідувати ітеративному, поетапному підходу, але субпідряднику, ймовірно, доведеться зробити процес прогнозним, вказавши кількість ітерацій та результати, пов'язані з кожною ітерацією, щоб конкурувати. Через це припущення щодо взаємодії з клієнтом, комунікації в команді, особистого спілкування, документації та змінних вимог можуть не виконуватися, коли робота передається на субпідряд у рамках проекту.

Як обговорювалося вище, перші три припущення передбачають, що розробники та клієнти знаходяться поруч, щоб вони могли взаємодіяти віч-на-віч, коли це необхідно. Можливо, не вдасться розмістити субпідрядників разом із розробниками та клієнтами. У цих випадках ті ж проблеми, що були виявлені для розподіленої розробки, існують і для субпідряду.

Вимагаючи від субпідрядників спільного розташування з основними розробниками та замовником, ці проблеми можна вирішити.

Як обговорювалося вище, припущення щодо документації стверджує, що документація (окрім фактичного програмного коду) повинна створюватися лише за необхідності. У субпідряді, як було описано для розподілених середовищ розробки, документація важлива, оскільки люди та команди, які не працюють разом щодня, повинні спілкуватися та надавати інформацію, щоб інші (інші субпідрядники, основні розробники, замовник тощо) могли взаємодіяти з тим, що було зроблено, та оцінювати його прийнятність у рамках проекту.

З огляду на більшу «відстань» між основними розробниками та субпідрядниками, а також між субпідрядниками та замовником, припущення про те, що документація не така вже й важлива, легко вважати недійсним.

Організація, що займається гнучкою розробкою, мало що може зробити

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		67

для вирішення цієї проблеми, окрім як збільшити обсяг своєї документації або вимагати від субпідрядників працювати разом з ними.

Припущення про зміну вимог стверджує, що вимоги завжди розвиваються. Однак субпідрядники зазвичай отримують тендер на розробку програмного забезпечення для фіксованого набору вимог.

Якщо вимоги часто змінюються, то й контракт має часто змінюватися, і це може призвести до значного збільшення витрат, оскільки в контрактах зазвичай зазначено, що за кожну зміну контракту стягуватиметься додаткова плата. Основа контракту, який використовують Agile-розробники, та основа контракту субпідрядників, принципово відрізняється, оскільки одна передбачає зміну вимог, а інша - фіксований набір заданих вимог.

Для вирішення цієї проблеми можливо укласти контракти, які надають субпідряднику певну гнучкість у способі розробки продукту в рамках часових та фінансових обмежень. Це, безумовно, можливо, якщо субпідрядник має хороший послужний список і компанія-підрядник може довірити йому розробку продукту, який відповідає потребам компанії-підрядника. Контракт, що підтримує гнучку розробку в середовищі субпідрядників, може складатися з двох частин:

- Фіксована частина: Ця частина визначає (1) рамки, що обмежують те, як субпідрядник вноситиме зміни до продукту (наприклад, критерії вартості та часу для прийняття або відхилення змін до Змінної частини (див. нижче) договору), (2) дії, які має виконати субпідрядник (наприклад, дії із забезпечення якості), та (3) вимоги, які слід вважати фіксованими, та результати, які необхідно доставити.

- Змінна частина: Ця частина визначає вимоги та результати, які можуть змінюватися в межах, визначених у Фіксованій частині. Ця частина може розвиватися в межах обмежень, визначених у Фіксованій частині. Під час підписання контракту слід включити опис пріоритетних результатів та вимог.

Великі команди часто мають багато підкоманд спеціалістів, і вони можуть знаходитися в різних географічно розподілених місцях. Великі команди зазвичай

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

зосереджуються на дуже великих проектах, де для вирішення проблем проекту потрібна велика кількість людських ресурсів. Через ці проблеми великі команди потребують більшої взаємодії між своїми членами та вищого ступеня зосередженості для управління ними. У цих середовищах припущення щодо взаємодії з клієнтами, спілкування в команді, особистого спілкування та документації можуть не виконуватися.

Розмір команд може обмежувати ефективність та частоту особистої взаємодії. Гнучкі процеси підтримують «управління в малому» тим, що механізми координації, контролю та комунікації застосовні до малих та середніх команд. У більших командах кількість ліній зв'язку, які необхідно підтримувати, може знизити ефективність таких практик, як неформальне особисте спілкування та оглядові зустрічі. Великим командам потрібні менш гнучкі підходи для вирішення проблем, характерних для «управління в цілому».

Немає багато чого, що можна зробити для вирішення цього припущення, окрім як спробувати мінімізувати розмір команди та максимізувати взаємодію, яка відбувається, водночас не дозволяючи обсягу взаємодії перевантажувати розробників та клієнта(ів).

Для великих команд неминуче потрібна більша кількість документації, просто для координації між великою кількістю членів команди. З огляду на їхню думку, що будь-яка документація, окрім коду, має бути мінімізована, гнучкі процеси розробки надають обмежену підтримку для розробки за участю великих команд.

Традиційні практики розробки програмного забезпечення, які наголошують на документації, контролі змін та архітектурно-орієнтованій розробці, більше застосовні для великих команд. Це не означає, що гнучкі методи не застосовні в таких середовищах. Для великих команд можуть бути можливості використовувати гнучкі практики, але ступінь можливої гнучкості може бути меншим, ніж у менших проектах. Наприклад, велика команда загалом може мати суворі вимоги до документації, але в рамках цього невеликі команди можуть застосовувати гнучкі методи розробки під час роботи над своїм проектом. Після

завершення проекту або через певні проміжки часу команда може документувати певні аспекти проекту, щоб відповідати вимогам великої команди. Це дозволить виконувати більшу частину роботи гнучким способом і лише в кінці (або в інші визначені моменти) створювати необхідну документацію.

Артефакти багаторазового використання – це код та інші компоненти (документи аналізу та проектування, шаблони тощо), які можна використовувати повторно з одного проекту в інший, повністю або принаймні в значній частині. Щоб створити компоненти, які можна використовувати повторно, необхідно враховувати загальну картину під час їх розробки, а не просто зосереджуватися на поточному застосунку. Які інші типи систем / застосунків можуть отримати вигоду від цього компонента? Скільки різних способів можна використовувати? Які вимоги предметної області, на відміну від просто цього застосунку в предметній області? Це лише деякі з питань, які необхідно поставити, думаючи про те, щоб зробити компоненти повторно використовуваними та більш універсальними. Під час розробки артефактів багаторазового використання припущення гнучкої розробки щодо документації, забезпечення якості, розробки для конкретного застосунку та постійного редизайну можуть бути недійсними.

Якщо документацію, окрім фактичного коду, мінімізувати, може бути важче визначити, коли і де певний артефакт можна використовувати повторно. Може знадобитися додаткова документація, щоб допомогти вказати на можливості повторного використання артефакту. Щоб гнучкі процеси підтримували розробку артефактів повторного використання, їм може знадобитися збільшити обсяг створеної документації.

Гнучкі процеси, такі як екстремальне програмування, зосереджені на створенні програмних продуктів, що вирішують конкретні проблеми. Розробка в "часі Інтернету" часто перешкоджає розробці узагальнених рішень, навіть коли очевидно, що це може принести довгострокові переваги. У такому середовищі розробку узагальнених рішень та інших форм програмного забезпечення повторного використання (наприклад, фреймворків проектування) найкраще вирішувати в проектах, які в першу чергу пов'язані з розробкою артефактів

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

повторного використання. Це відокремлення середовища розробки, специфічного для продукту, від середовища розробки артефактів повторного використання є основною особливістю фреймворку, орієнтованого на повторне використання, під назвою " *Фабрика досвіду*", розробленого дослідниками з Університету Меріленду в Коледж-Парку [5]. Широке застосування артефакту повторного використання вимагає, щоб процес, який використовується для створення артефакту, наголошував на контролі якості, оскільки вплив низької якості (зокрема, серйозних помилок) такий же широкий, як і кількість програм, які повторно використовують артефакт. З іншого боку, бажана своєчасна розробка артефактів повторного використання.

Безперервний редизайн ускладнюється, коли не розробляються артефакти, специфічні для програми. Можливість отримання відгуків від клієнтів зменшується, а отже, зменшуються й покращення якості та дизайну. Щоб вирішити цю проблему, гнучкі розробники повинні запровадити спеціальні процеси, призначені для отримання такого типу відгуків, щоб покращити дизайн та якість артефактів повторного використання.

Здається очевидним, що гнучка розробка не дуже добре підходить для створення артефактів багаторазового використання. Однак, за умови ретельної уваги та внесення деяких ключових коригувань до гнучких процесів, як згадувалося вище, можливо успішно адаптувати та застосувати гнучкі процеси до розробки артефактів багаторазового використання.

Програмне забезпечення, критично важливе для безпеки, – це програмне забезпечення, яке може загрожувати життю, здоров'ю чи безпеці людей, якщо якість програмного забезпечення не є надзвичайно високою. Деякі приклади включають програмне забезпечення для управління авіацією та програмне забезпечення/програмне забезпечення для керування рентгенівськими апаратами. У таких середовищах важливо знати, що програмне забезпечення пройшло ретельне тестування та розроблено таким чином, щоб гарантувати відсутність збоїв, які впливають на здатність правильно та безпечно використовувати та керувати обладнанням. Неприпустимо, щоб машині

дозволяли видавати дози рентгенівського опромінення, які можуть бути смертельними для пацієнта, який їх отримує, або щоб пілот не міг керувати літаком, наприклад, через збій програмного забезпечення. У таких ситуаціях припущення гнучкої розробки щодо документації, забезпечення якості та безперервного редизайну можуть бути недійсними.

Формальна специфікація, ретельне тестування та інші методи формального аналізу та оцінки, що входять до підходів до програмної інженерії, забезпечують надійніші, але також і дорожчі механізми для розробки програмного забезпечення, критично важливого для безпеки або бізнесу. Ці підходи можуть надійніше «гарантувати», що відповідні тести були проведені, а код проаналізовано, щоб розробники та користувачі були впевнені в безпеці та надійності системи.

Застосування деяких методів гнучкого оцінювання до такого програмного забезпечення також може бути корисним. Наприклад, (1) підходи «спочатку тестування» вимагають визначення модульних тестів перед написанням коду, (2) раннє створення робочого коду, підтримуване ітеративною, інкрементальною структурою процесів гнучких процесів, підтримує дослідницьку розробку критичного програмного забезпечення, в якому вимоги не є чітко визначеними, та (3) парне програмування може бути ефективним доповненням до формальних оглядів.

Отже, можна припустити, що гнучка та формальна розробка програмного забезпечення не є несумісними, але їх можна поєднувати за потреби: формальні методи можуть використовуватися в поєднанні зі гнучкими процесами для обробки критичних частин програмного забезпечення з метою підвищення якості та впевненості.

Велике, складне програмне забезпечення - це програмне забезпечення, яке містить великі обсяги коду (сотні тисяч, мільйони тощо рядків) та/або може включати дуже складні взаємозв'язки між різними частинами системи для забезпечення цілісності даних та надійної взаємодії всіх частин системи та їхньої роботи належним чином. Розробка великого, складного програмного

забезпечення зазвичай вимагає вищого ступеня управлінського контролю та більшої кількості більш «формалізованих» процесів, щоб переконатися, що все підходить, працює разом і працює надійно. Припущення гнучкої розробки щодо документації, забезпечення якості, ітерацій та безперервного редизайну можуть бути недійсними в цих ситуаціях.

Як обговорювалося вище щодо великих команд розробників, під час розробки великого, складного програмного забезпечення, ймовірно, існує підвищена потреба в документації. Це необхідно для простого документування більшого набору вимог, функцій та проектних рішень, а також для забезпечення бази знань для більших команд, які, ймовірно, працюватимуть над такими системами. Зосередження майже виключно на коді для документації може призвести до серйозного нерозуміння системи та більш складного завдання навчання нових членів команди під час та після завершення проекту. Якщо Agile-розробники свідомо докладуть зусиль для документування ключових рішень, проектів тощо, то цього обмеження можна буде уникнути.

Так само, припущення, що неформальне тестування та огляди можуть забезпечити необхідний рівень якості у великих складних системах, ймовірно, є хибним. Якщо застосовується гнучкий підхід створення тестів перед написанням коду (спочатку тестування), а процес розробки цих тестів є ретельним та добре задокументованим, тоді проблем може не виникнути. Однак це необхідно забезпечити для підтримки якості у великих складних системах.

Припущення про ітерації також може бути недійсним під час розробки великого, складного програмного забезпечення, оскільки можуть існувати системи, в яких функціональність настільки тісно пов'язана та інтегрована, що розробка програмного забезпечення поступово може бути неможливою. У цих випадках все ще можна використовувати ітеративний підхід, за якого код створюється на кожній ітерації, але код, створений на кожній ітерації, включатиме всі частини в різних станах незавершеності.

Зрештою, припущення, що рефакторинг коду усуває необхідність проектування з урахуванням змін, може не відповідати, зокрема, великим

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

складним системам. У такому програмному забезпеченні можуть бути критичні архітектурні аспекти, які важко змінити через критичну роль, яку вони відіграють в основних послугах, що пропонуються системою. У таких випадках вартість зміни цих аспектів може бути дуже високою, тому варто докласти додаткових зусиль, щоб передбачити такі зміни на ранній стадії. Залежність від рефакторингу коду також може бути проблематичною для таких систем. Складність і розмір такого програмного забезпечення можуть зробити суворий рефакторинг коду дорогим і схильним до помилок. Моделі можуть відігравати тут важливу роль, особливо якщо існують інструменти для генерації значних частин коду з моделей. Такий погляд на моделі як на центральні артефакти для систем, що розвиваються, лежить в основі підходу до модельно-орієнтованої архітектури (MDA) Групи управління об'єктами (OMG).

3.3 Оцінка ефективності моделей та інтерпретація отриманих результатів

У таблиці наведено 37 вибраних статей, а на рис. 3а зображено розподіл за роками публікацій.. Хоча маніфест Agile вже був опублікований у 2001 році, дослідження, що пов'язують моделювання бізнес-процесів та Agile, публікувалися інтенсивніше після 2010 року, з піком у 2016 році. Це можна пояснити критикою, яка виникла щодо Agile-розробки на початку років з боку деяких практиків [24], та певним інтересом з боку прихильників Agile на той час до того, щоб його не порівнювали з традиційними методологіями [17]. Ці два явища могли вплинути на інтерес дослідників та практиків до використання підходів, тісно пов'язаних з традиційними методологіями, таких як моделі бізнес-процесів.

З відібраних статей 62% (23 дослідницькі роботи) отримали *дуже високі* або *високі* оцінки якості. Ми спостерігаємо, що статті з високими оцінками якості часто були написані між 2015 і 2017 роками, що також є періодом піку кількості публікацій про інтеграцію моделей бізнес-процесів зі гнучкою розробкою

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

програмного забезпечення. Вибір гнучкої методології в первинних дослідженнях представлено на рис. 3b. Більшість статей (57%) згадують свою спрямованість на гнучкі методології загалом, не вказуючи якусь конкретну методологію. Інші 43% статей використовують три методології, розподілені у таких відсотках: Scrum (24%), XP (12%) та Kanban (7%).

Другою за популярністю основою моделювання є UML (15%). Хоча UML не був спеціально створений для моделювання бізнес-процесів, у різних програмних проектах процеси моделюються за допомогою діаграм діяльності або діаграм послідовностей.

Ми зафіксували контекст промислового застосування наступним чином. У 21 статті (57%) не повідомляється про контекст промислового застосування. Решта 16 (43%) статей охоплюють різноманітні області застосування, як показано в таблиці 3.2.

На рис. 3.5 показано, що твердження (27%) та тематичні дослідження (24%) були найчастіше використовуваними методами валідації (відповідно до типології) у вибраних первинних дослідженнях.

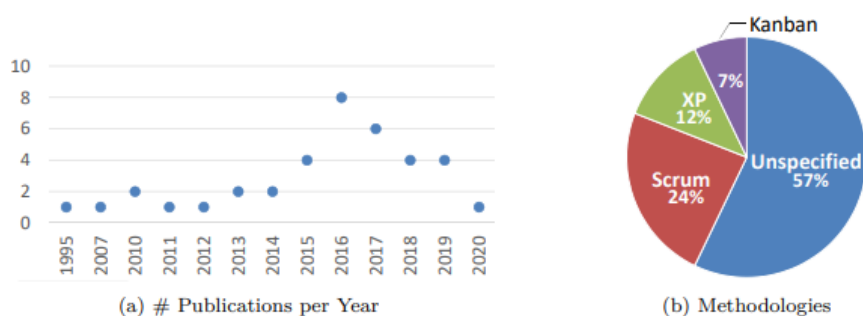


Рисунок 3.5 - Дослідження, класифіковані за роком публікації та методологіями розробки програмного забезпечення.

Той факт, що значний відсоток досліджень валідується за допомогою тверджень (тобто *ad hoc* валідація, де приклади можуть сприяти результатам), є слабкістю. Однак ми також можемо оцінити це як можливість, оскільки це являє

собою перший крок до майбутніх експериментів та промислового застосування. Те саме стосується і досліджень без валідації (19%).

У деяких вибраних дослідженнях (11%) метод валідації використовувався для оцінки завершених проєктів та визначення отриманих уроків. Наприклад, цей тип оцінювання застосовується. Вони використовують процес зворотного проєктування над поставленими програмними продуктами, щоб визначити популярні методології розробки програмного забезпечення в ігровій індустрії.

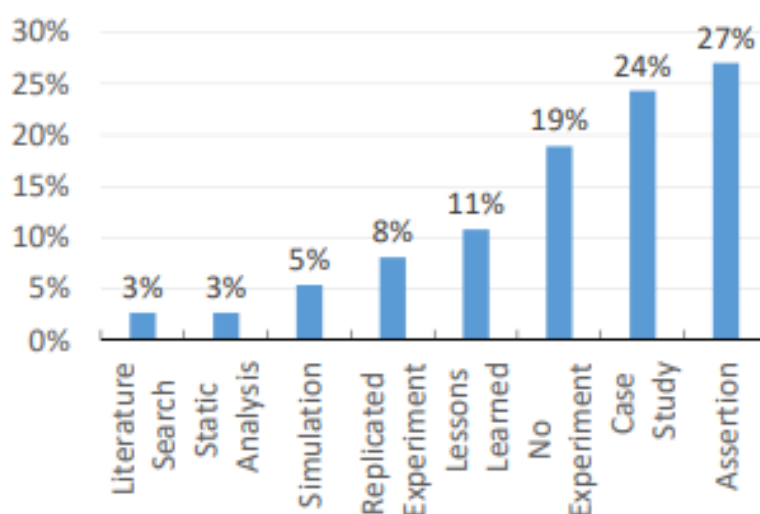


Рисунок 3.6 Типи перевірки

Незважаючи на невисокий відсоток застосування, реплікація експериментів (8%) також була методом оцінювання, що застосовувався в деяких вибраних дослідженнях. У цьому типі оцінювання один і той самий експериментальний дизайн використовувався кілька разів, можливо, з незначними змінами в кожному з виконань. Прикладом такого методу оцінювання є дослідження у роботі було проведено експеримент, щоб оцінити, чи зацікавлені сторони краще розуміють залежності між користувацькими історіями з допомогою чи без допомоги бізнес-процесних моделей.

Щодо типу програмного продукту, пов'язаного з вибраними дослідженнями, було виявлено, що більшість із них не згадують конкретний тип продукту (50%).

Таблиця 3.2:

Типи артефактів у Scrum

Артефакт	Опис	Призначення
Беклог продукту	Список усіх функцій, вимог та виправлень, необхідних для продукту.	Визначення обсягу робіт на майбутнє.
Беклог спринту	Список завдань, обраних командою для виконання протягом поточного спринту.	Планування роботи на короткий період.
Інкремент продукту	Сума всіх завершених елементів беклогу за результатами спринту.	Демонстрація готової до використання частини продукту.

На противагу цьому, інші дослідження вказують на веб- (40%), мобільні (8%) та автономні (3%) додатки, задіяні в їхній роботі.

У наступних розділах ми представляємо результати щодо дослідницьких питань (RQ1 - RQ3).

У таблиці 3.2 показано частоту різних застосувань моделей бізнес-процесів у гнучкій розробці програмного забезпечення згідно з літературою. Зауважте, що зв'язок між використанням та первинними дослідженнями є зв'язком «багато до багатьох». Далі ми детально представляємо ці використання.

U1 (*аналіз та проектування системи з нефункціональної точки зору, але з урахуванням потреб бізнесу*) є найчастішим використанням у нашому дослідженні. Підкреслюють, що проектування системи може включати шаблони, виявлення функціональних та якісних вимог, а отже, функціональні та нефункціональні проекти. Функціональні вимоги визначають поведінку системи, а нефункціональні вимоги - як система повинна бути розроблена для виконання аспектів якості, які не охоплюються функціональними вимогами. Останні здебільшого пов'язані з архітектурним проектуванням.

U2 (*генерація прототипів програмного забезпечення*) та U3 (*моделювання процесу, який слід дотримуватися під час впровадження системи*) були ідентифіковані однаково часто в нашому аналізі. U2 зосереджується на генерації вихідного коду з моделей процесів та створенні

відповідних інтерфейсів користувача. Стверджують, що, незважаючи на високі початкові зусилля, пов'язані зі створенням моделі процесу, перетворення на вихідний код робить моделювання дуже привабливим завдяки економії часу. Цю точку зору поділяють кілька інших досліджень. U3, у свою чергу, підкреслює, що моделі бізнес-процесів використовуються для визначення того, чого різні зацікавлені сторони повинні досягти крок за кроком до впровадження кінцевого програмного продукту. Приклад такого використання представлено, які створили інтерактивний інструмент, де розробники навчаються за допомогою моделей процесів створювати програмне забезпечення відповідно до Scrum.

Далі у списку йде U4 (*підтримка валідації та тестування системи*). Зазвичай валідація та тестування вимагають зустрічей з клієнтами. Корисно мати моделі бізнес-процесів як підтримку. Наприклад, створюють автоматизовані тести на основі моделей процесів, що скорочує час та спрощує тестування для розробників.

U5 (*створення загального уявлення про обсяг проекту*) наполегливо рекомендується Інститутом управління проектами, а практики вважають його «фундаментальним кроком у проекті розробки програмного забезпечення». Враховуючи, що моделі бізнес-процесів фіксують те, що потрібно розробити або що вже використовується, вони допомагають зацікавленим сторонам зрозуміти обсяг того, що має забезпечити процес розробки програмного забезпечення.

Згідно з частотою, наступним використанням є U6 (*співпраця із зацікавленими сторонами у визначенні вимог*). Наявність такого інструменту, як моделі процесів, який, наприклад, допомагає виявити вимоги на зустрічі із зацікавленими сторонами, дозволяє уникнути пропуску відповідних вимог.

U7 (*замінити Випадки використання/Історії користувача*) стосується документації, яка зазвичай використовується в гнучких проектах розробки програмного забезпечення. Стверджують, що такі артефакти, як історії користувача, не допомагають безпосередньо уникнути проблем комунікації під час процесу розробки програмного забезпечення. Замість цього вони пропонують використовувати графічні представлення, такі як діаграми BPMN.

Інші дослідники розглядають цю пропозицію як варіант, але також оцінюють можливість доповнення історій користувача моделями бізнес-процесів, що є U10 (*доповнення Випадки використання /Історії користувача*).

Шаблони створюються для зменшення обсягу роботи в повторюваних або подібних процесах. Гнучка розробка програмного забезпечення має на меті мінімізувати час розробки, і U8 (*уникнення повторення та переробки шляхом створення шаблонів спільних процесів*) спрямована на досягнення цієї мети. Наприклад, пропонується повторне використання специфікацій вимог, створених за допомогою діаграм BPMN, у проектах з подібним або однаковим обсягом.

U9 (*аналіз та проектування системи з функціональної точки зору*) та U1 не були інтегровані, незважаючи на однакову кінцеву мету – проектування системи. Причиною їхньої відсутності інтеграції є різниця в обсязі проектування. У той час як U1 зосереджений на нефункціональних аспектах (наприклад, архітектура, інтерфейси), U9 орієнтований на проектування функціональних можливостей (наприклад, інтеграція моделей бізнес-процесів з моделями рішень для чіткого вираження того, як система повинна діяти). Так само U10 (*доповнення варіантів використання/історій користувачів*) та U7 також не були інтегровані. Хоча варіанти використання або історії користувачів згадуються для обох, U7 прагне повністю замінити їх моделями бізнес-процесів. Навпаки, U10 не пропагує повну заміну, а їх поєднання.

U11 (*підтримка діяльності з визначення пріоритетів, оцінювання та відстеження залежностей*) зосереджена на розумінні зв'язків між різними функціональними можливостями системи та, водночас, на тому, як ці функціональні можливості пов'язані з діяльністю бізнес-процесів. Це розуміння дозволяє визначити пріоритети необхідних розробок або завдань та генерувати точні оцінки (наприклад, часу, вартості та обсягу). Наприклад, пропонують метод представлення залежностей між користувацькими історіями шляхом пов'язування їх з діяльністю в моделях бізнес-процесів.

Інше використання, яке частково стосується U1, це U12 (*підтримка*

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

аспектів безпеки в розробці програмного забезпечення), оскільки безпека є нефункціональною вимогою, яка визначає аспекти якості системи. Тим не менш, вона була окремо розглянута, оскільки були знайдені дослідницькі роботи, які спеціально оцінювали, як підтримувати функції безпеки в гнучких проектах розробки програмного забезпечення. Вважають, що безпека є критичним питанням, особливо для складних гнучких проектів, і це може бути підтримано моделями бізнес-процесів.

Три з цих застосувань мали унікальні характеристики, але вони не часто згадуються. U13 (*підтримка діяльності з технічного обслуговування*) стосується використання моделей бізнес-процесів для діяльності з технічного обслуговування, включаючи виправлення помилок, відповіді на запитання клієнтів або розробку нових функцій, узгоджених з існуючими. Пропонують використовувати моделі процесів, які вже використовувалися для виявлення, документування та перевірки вимог у діяльності з технічного обслуговування.

U14 (*зворотне проектування для виявлення процесу, який був використаний для створення системи*) – це ще одне конкретне застосування. Зворотне проектування – це процес, який дозволяє визначити компоненти, зв'язки та процеси, що були реалізовані. У цьому сценарії використовувалися моделі бізнес-процесів для визначення деталей застосованої методології розробки програмного забезпечення. Це особливо корисно, враховуючи, що на практиці гнучкі методології часто адаптуються до певного середовища (наприклад, різних проектів). Тому, щоб покращити процес створення систем у компанії, необхідно визначити уявлення про те, що було зроблено в кожній ситуації.

Нарешті, ми визначаємо U15 (*посилення співпраці між зацікавленими сторонами*). На відміну від інших застосувань, де співпраця має певний фокус (наприклад, співпраця із зацікавленими сторонами у визначенні вимог (U6)), U15 пов'язана із загальною співпрацею зацікавлених сторін з цілісної точки зору. Пропонують використовувати моделі процесів як фундаментальну основу для всього процесу розробки програмного забезпечення. У їхньому підході різні

зацікавлені сторони (наприклад, керівники проектів, розробники та архітектори програмного забезпечення) взаємодіють один з одним через програмний застосунок, який використовує моделі процесів для підтримки потреб комунікації та координації.

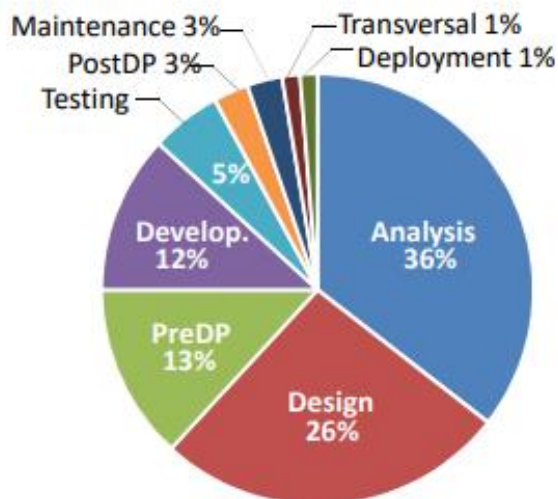


Рисунок 3.7 - Використання моделей бізнес-процесів у гнучкій розробці програмного забезпечення за етапами

Проект гнучкої розробки програмного забезпечення включає більше видів діяльності, ніж передбачено циклом розробки програмного забезпечення. Не вдаючись у складні деталі управління проектами та його численні групи процесів і області знань (наприклад, PMI), ми спираємося тут на три фундаментальні фази проекту гнучкої розробки програмного забезпечення: передрозробка, розробка та післярозробка. Крім того, існують поперечні види діяльності, які можуть впливати на всі фази. На рис. 3.7 показано, якою мірою моделі бізнес-процесів використовуються на різних фазах проекту гнучкої розробки програмного забезпечення згідно з дослідницькою літературою.

Фаза *перед розробкою* (PreDP) складається з заходів, що виконуються до розробки. Ці заходи включають сприяння або підготовку роботи що має бути зроблено на наступних етапах, наприклад, визначення обсягу проекту або навчання зацікавлених сторін певній методології. Фаза *розробки* включає цикл

розробки програмного забезпечення (тобто аналіз, проектування, впровадження, тестування, розгортання, обслуговування). Ці дії включені до методологій програмного забезпечення, але з різними обсягами. Фаза *після розробки* (PostDP) зосереджена на діях, які можна вжити після завершення розробки. Дії, що вживаються на цій фазі, можуть служити технічним потребам або потребам управління проектом. Прикладом є виконання оцінок після розробки для винесення висновків. Нарешті, *поперечні* дії – це ті, які можуть відбуватися на будь-якій з фаз. Ці дії пов'язані із загальними темами, такими як безпека.

Загалом, ми виявили, що моделі бізнес-процесів використовуються на всіх фазах процесу гнучкої розробки програмного забезпечення, але переважно на його початку. Рис. 3.7 показує, що аналіз вимог (36%) та проектування (26%) найчастіше використовуються. У таблиці наведено додаткові деталі щодо зв'язків між використаннями (U1-U15) та згаданими раніше фазами. Числа в кожній клітинці відносяться до сегментів (S_m), які належать до використання (U_p) та відповідних фаз. Зауважте, що зв'язок між сегментами та фазами є зв'язком «багато до багатьох». Ми спостерігаємо, що найчастішими зв'язками є U1 (аналіз та проектування системи з нефункціональної точки зору, але з урахуванням бізнес-потреб) у проектуванні, U2 (генерація прототипів програмного забезпечення) у розробці та U3 (моделювання процесу, який слід дотримуватися під час впровадження системи) у PreDP.

На етапі аналізу ми спостерігаємо, що моделі бізнес-процесів в основному використовуються для створення загального уявлення про обсяг проекту (U5), як інструмент для співпраці для визначення вимог (U6) та для заміни поширених користувацьких історій (U7). На етапі проектування моделі бізнес-процесів застосовуються для проектування системи з функціонального та нефункціонального підходу (U1 та U9).

Таблиця 3.3:

Ключові зустрічі (Події) у Scrum

Подія	Мета	Тривалість (стандарт)
Планування спринту	Визначити ціль спринту та обрати завдання з беклогу.	До 8 годин (на місячний спринт).

Щоденний Scrum (Daily)	Синхронізація планів команди на наступні 24 години.	15 хвилин.
Огляд спринту (Review)	Демонстрація результатів стейкхолдерам та збір відгуків.	До 4 годин (на місячний спринт).
Ретроспектива спринту	Аналіз робочих процесів команди та пошук шляхів до покращення.	До 3 годин (на місячний спринт).

У таблиці також показано зв'язки, які є менш поширеними, ніж три попередні приклади, а саме: U3 (*моделювання процесу, якого слід дотримуватися під час впровадження системи*) у розділі «Розгортання» та U11 (*підтримка дій з визначення пріоритетів, оцінки та відстеження залежностей*) у розділі «Аналіз та проектування». Вони використовують діаграми бізнес-процесів для повідомлення альтернативних сценаріїв DevOps різним зацікавленим сторонам та для аналізу їхніх переваг і недоліків. U11 стосується розділу «Аналіз та проектування». Стверджують, що завдяки інтеграції моделей бізнес-процесів та історій користувачів потік і залежності між історіями користувачів стають краще видимими. Ця інтеграція представлена графічно шляхом розміщення ідентифікаційного коду історії користувача (наприклад, US-19) поруч із відповідною діяльністю на діаграмі процесу.

Ми провели фокус-групу з фахівцями. Її результатом є категоризація визначених застосувань моделей бізнес-процесів відповідно до різних потреб у проектах гнучкої розробки програмного забезпечення. У таблиці 3.3 показано результат за чотирма категоріями: управління, команда, технології та процеси. Відсотки, представлені за використанням, відображають рівень, на якому фахівці вважають, що відповідне використання може підтримувати кожну категорію. Як описано це оцінюється з точки зору аспектів, які потребують підтримки в проектах гнучкої розробки програмного забезпечення. Ці аспекти є частиною опису кожної категорії та представлені в таблиці 3.3. Результати були зважені за використанням, щоб зробити їх порівнянними між чотирма категоріями, і ми виділили зеленим кольором категорію, де кожне використання може мати найбільший вплив.

Більшість із 15 застосувань були класифіковані за кількома категоріями. Іншими словами, ми часто знаходили використання моделей бізнес-процесів, які

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		83

можуть бути корисними для кількох аспектів гнучкої розробки програмного забезпечення одночасно. Однак рівень впливу кожного з цих застосувань у кожній категорії відрізняється залежно від характеру використання. Наприклад, U1 (*аналіз та проектування системи з нефункціональної точки зору, але з урахуванням потреб бізнесу*) класифікується як використання, яке переважно корисне для процесів (40%) гнучкої розробки програмного забезпечення, але водночас має потенціал позитивно впливати на управлінські (38%), командні (13%) та технологічні (9%) аспекти гнучкої розробки. Як інший приклад, U4 (*підтримка валідації та тестування системи*) також впливає на всі чотири категорії, але результати вказують на те, що менеджмент (41%) є основним бенефіціаром.

Як і очікувалося, враховуючи фокус нашого дослідження та те, що моделі бізнес-процесів стосуються процесів, для 10 випадків використання (67%) категорія «процес» має найвищий відсоток. Другою за величиною категорією є управління (4 способи використання, 27%). Категорією, в якій більшість способів використання мають найменший вплив, є технології.

Щодо фокус-групи з практиками, деякі варті уваги коментарі вплинули на результати, зображені в Таблиці 3.4. Наприклад, для категорії «Процес» практики підкреслюють роль моделей бізнес-процесів у визначенні потоку інформації, підтримці специфікації вимог та сприянні складним проектам. Хоча існують програми, які дозволяють керувати вимогами або розробкою продуктів.

Таблиця 3.4:

Переваги впровадження Agile-підходу

Перевага	Опис	Вплив на проєкт
Прозорість	Усі учасники бачать реальний стан справ у реальному часі.	Зменшення ризиків та краща керованість.
Адаптивність	Можливість швидко реагувати на зміни ринку або вимог.	Вища конкурентоспроможність продукту.
Якість	Раннє тестування та постійні перевірки.	Менша кількість багів у фінальній версії.
Залученість клієнта	Клієнт постійно бере участь у процесі розробки.	Відповідність продукту очікуванням користувача.

Експерти зазначають, що визначити залежності між функціональними можливостями, просто підтримуючи зв'язок між історіями користувачів у цих додатках, не є простим завданням. З цієї причини фахівці повідомляють не лише про використання цих додатків, а й про створення моделей бізнес-процесів. «Вони [моделі процесів] полегшують спільне визначення та відстеження того, що ми збираємося робити з функціональної та нефункціональної точки зору».

Фахівці повідомляють, що вони використовують моделі процесів для різних цілей та рівнів деталізації. Залежно від проекту та його характеристик (наприклад, часу, типу залучених людей, складності), рівень деталізації варіюється. Один фахівець сказав: «Ми використовуємо їх моделі бізнес-процесів для доповнення історій користувачів. Ми вважаємо практичним поєднувати детальне візуальне представлення вимог з текстовим описом». В іншому прикладі: «У нашій компанії, як правило, для кожного нового проекту/продукту середнього розміру, який ми починаємо, ми створюємо діаграму високого рівня, що відображає обсяг продукту. Очевидно, що ця діаграма з часом змінюється ». Цей коментар підтримали інші, які додали, що для них «ці діаграми виступають візуальним представленням продукту або дорожньою картою проекту».

Фахівці неодноразово згадували, як вони повторно використовують створені моделі для інших функціональних можливостей, проектів чи продуктів. Один фахівець зазначив, що *«так само, як існують шаблони проектування для управління конкретними та поширеними проблемами в програмному забезпеченні, також існують визначені процеси, які вирішують повторювані вимоги »*. Визначення того, що можна повторно використовувати з моделей процесів, також допомагає проаналізувати, що можна повторно використовувати з коду, пов'язаного з моделями процесів.

Оцінюючи, чи може використання (U1 - U15) підтримувати різні аспекти гнучкої методології (Agile) у категорії «Менеджмент», фахівці наголосили, як моделі бізнес-процесів допомогли їм приймати рішення, залучати зацікавлені -

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		85

сторони та налагоджувати ефективну комунікацію. Щодо прийняття рішень, фахівці згадали про необхідність постійних змін у гнучкій методології, що неминуче призводить до постійних змін у загальних цілях та стратегіях. Це означає, що *«відповідні рішення мають прийматися за короткі періоди часу, і нам потрібні інструменти, які допомагають нам це робити»*. Фахівець зазначає, що використання моделей бізнес-процесів допомогло їм, серед іншого, виявити та вжити заходів щодо неефективних взаємодій, можливого впливу змін та пріоритетів.

Щодо залучення зацікавлених сторін та ефективної комунікації, практики зазначають, що вони розглядають моделі процесів як *«практичний механізм для того, щоб усі говорили однією мовою»*. Крім того, вони погоджуються, що наявність моделі процесу, на яку можна посилатися під час розмови з клієнтами або на внутрішніх зустрічах, полегшує отримання та надання зворотного зв'язку. Це пояснюється тим, що *«складні ідеї легше відстежувати»*, *«розмови залишаються зосередженими»*, а також є *«візуальний компонент, який допомагає людям зрозуміти, пояснити та запам'ятати, що потрібно зробити»*. Як і в інших типах проектів, ефективна комунікація є вирішальним елементом, але її актуальність виходить на новий рівень у гнучких методологіях. Наприклад, розуміння потреб клієнтів якомога раніше та отримання постійного зворотного зв'язку є важливим для визначення майбутніх дій. Усі практики поділилися позитивним досвідом, пов'язуючи моделі бізнес-процесів з покращенням комунікації та залучення. Досвід, яким вони поділилися, включає наступне:

«Після створення моделей бізнес-процесів на зустрічах з одним із наших клієнтів, клієнт зрозумів, що компанія пропонує для складного нового функціоналу. Окрім швидшого отримання схвалення на подальшу роботу над проектом, ми також виявили, що клієнт та внутрішня команда почали обмінюватися ідеями та змінами, посиляючись на конкретні дії в моделі».

«Ми використовуємо моделі процесів, щоб пояснити новим членам команди, як працює процес розробки програмного забезпечення в компанії. Це

було значним покращенням порівняно з текстовим посібником, який ми їм давали раніше. Тепер вони швидше впроваджуються в процес».

«Під час планування спринту ми часто звертаємося до моделі процесу, яку створили разом бізнес- та технічна команда. Це допомагає всім зрозуміти складність змін».

Незалежно від аспекту чи категорії, яку слід підтримувати в гнучкій розробці програмного забезпечення за допомогою моделей бізнес-процесів, практики повторили важливість «збереження простоти». Один практикуючий спеціаліст зазначив: «воно моделювання бізнес-процесів корисне до тих пір, поки ми не ускладнюємо його деталями нотації». Хоча нотації, такі як BPMN, вважаються більш зрозумілими для різних зацікавлених сторін, ніж, наприклад, UML, існують також проблеми зі складністю нотації для певних груп. Наприклад, «необов'язково суворо використовувати правильний шлюз, коли ми створюємо модель з клієнтом». Інші коментарі, що підтверджують цю ідею, представлені нижче:

«Я не надаватиму всіх деталей, необхідних для створення прийняттого коду. Це забагато зусиль, і вони того не варті. Однак, це гарна відправна точка для створення основи коду».

«В Agile зміни - єдина константа. Це також стосується моделей, які ми використовуємо. Може статися так, що нам доведеться змінити всю модель з тижня на тиждень. То чому ми повинні витратити зайвий час на її деталізацію?»

Було погоджено, що визначення того, що є «простим» у моделюванні, залежить від кожної ситуації. У деяких випадках може бути достатньо «використати основні елементи нотації для створення високорівневої моделі на аркуші паперу». В інших випадках це може означати «створення моделей, які будуть оновлюватися, мати версії та бути пов'язаними з кодом». Підсумовуючи, «простий» - це те, що відповідає потребам команди, залишаючись ефективним.

Загальні результати цього дослідження показують, що моделі надають різні переваги для розробки програмного забезпечення та підкреслюють ці переваги в контексті гнучкої розробки програмного забезпечення. Всупереч

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

очікуванню, моделі бізнес-процесів використовуються в проектах гнучкої розробки програмного забезпечення для різних цілей. Наші висновки щодо *RQ1* показують 15 застосувань моделей бізнес-процесів у гнучкій розробці програмного забезпечення. Оскільки це дослідження є першим, яке надає інтегрований погляд на використання моделей бізнес-процесів у гнучкій розробці програмного забезпечення, неможливо безпосередньо порівняти цей список застосувань з іншими дослідженнями. Однак, порівняння їх з дослідженнями, в яких моделювання та гнучка розробка програмного забезпечення інтегровані, можливе. Попередником цього типу інтеграції, який він називає гнучкою розробкою на основі моделей. Він пропонує використовувати високорівневі моделі для визначення архітектури системи, а потім як вхідні дані для інших артефактів, таких як вихідний код. Хоча його робота не зосереджена безпосередньо на моделях бізнес-процесів, у ній стверджується, що основне використання моделювання в гнучкій розробці програмного забезпечення зосереджено на аналізі та проектуванні системи. У цьому ж напрямку інші дослідники також наголошують на ролі моделей у цих двох сферах, незалежно від використаної методології. Наприклад наголошують на актуальності моделей для завдань аналізу та проектування, таких як «виявлення вимог», «уточнення вимог», «визначення дизайну», «декомпозиція дизайну» та «впровадження системи». Ці твердження резонують з нашими висновками. Враховуючи, що моделювання загалом та моделювання бізнес-процесів, зокрема, широко використовуються в діяльності, пов'язаній з фазами аналізу та проектування в традиційних методологіях не було несподіваним зробити подібні спостереження для гнучких проектів, як це представлено в *RQ2*. Це викликає питання про те, чому це так. Можна припустити, що це може бути пов'язано зі складністю завдань на цих двох фазах [6]. У цьому сценарії наявність візуального представлення допомагає впоратися з неоднозначністю, непорозуміннями та пропущеними деталями [7].

Хоча не дивно, що моделі бізнес-процесів широко використовуються на ранніх етапах циклу розробки, несподівано виявити використання моделей

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		88

бізнес-процесів на всіх етапах гнучкого проекту розробки програмного забезпечення. Це демонструє потенціал моделей бізнес-процесів у таких типах проектів і вимагає більш детального дослідження застосування моделей бізнес-процесів у реальних проектах. Крім того, результати RQ3 також дозволяють розглядати моделі бізнес-процесів як перспективний механізм для підтримки гнучкої розробки програмного забезпечення. Ми виявили, що переважна більшість із 15 застосувань належать до кількох категорій (управління, команда, технології, процеси), що означає, що більшість визначених застосувань можуть допомогти вирішити численні складні аспекти гнучкої розробки програмного забезпечення. На думку практиків, це можливо головним чином тому, що моделі процесів використовуються як інструмент комунікації, а комунікація є критичним елементом у кожному аспекті гнучкого проекту. Водночас стверджують, що комунікація є однією з найважливіших проблем, що впливають на гнучкі проекти. Проблеми комунікації впливають на рівень невдач цього типу проектів. Тому альтернативи, що підтримують та сприяють комунікації, є цінними активами для гнучких проектів розробки програмного забезпечення. Однак, як наголошували практики в цьому дослідженні, вважається, що корисність моделей процесів у гнучкому методології полягає в застосуванні практичного підходу. Це означає, що такі аспекти, як рівень деталізації або суворе дотримання нотації, залежатимуть від характеристик проекту, але завжди слід намагатися зберегти моделі та їх застосування максимально «простими», наскільки це необхідно. Ці твердження схожі на ті.

Щодо RQ3, було виявлено, що процеси та управління є двома сферами, де моделі бізнес-процесів можуть принести найбільшу користь у проектах гнучкої розробки програмного забезпечення. Ці результати та їх зв'язок з роботою підкреслюють, що моделі бізнес-процесів можуть бути цінним інструментом для вирішення проблем гнучкої розробки програмного забезпечення. Пріоритезують проблеми гнучкої розробки програмного забезпечення відповідно до їхньої серйозності, а управління - це категорія, де виявляється більше проблем з високим пріоритетом. Майбутні дослідження,

наприклад, дослідження, яке ранжує проблеми. [39], за ступенем підтримки моделей бізнес-процесів, може допомогти глибше зрозуміти цей взаємозв'язок.

Співпраця між моделюванням та гнучкою розробкою програмного забезпечення не була широко досліджена в дослідницькій сфері; і коли вона вивчалася, то зазвичай обговорювалося, як Agile може підтримувати моделювання. Подібні спостереження можна зробити щодо моделювання бізнес-процесів та гнучкої розробки програмного забезпечення. Хоча різні дослідницькі роботи показують, що моделі бізнес-процесів використовуються в проектах розробки програмного забезпечення на різних рівнях, літератури, яка явно оцінює інтеграцію моделей бізнес-процесів та гнучкої розробки програмного забезпечення, недостатньо. Дослідники наголосили на необхідності зосередити увагу на моделюванні в Agile [14, 32]. Оскільки наша робота є першою, яка надає цілісне уявлення про те, як моделі бізнес-процесів є та можуть бути використані в проектах гнучкої розробки програмного забезпечення, ми вважаємо, що вона має різні наслідки для дослідників та практиків.

По-перше, існує кілька можливостей для вторинних досліджень. Ці дослідження, наприклад, можуть заглибитися в додаткові деталі щодо загального взаємозв'язку між моделями бізнес-процесів та гнучкою розробкою програмного забезпечення. На основі наших результатів ми можемо вивести наступні теми навколо відкритих дослідницьких питань щодо інтеграції моделей бізнес-процесів та гнучкої розробки програмного забезпечення: (i) визначення та аналіз організаційних факторів, які фахівці оцінюють, приймаючи рішення про використання моделей бізнес-процесів, як описано в цій роботі, (ii) конкретні рекомендації щодо того, як моделі бізнес-процесів можна використовувати для вирішення проблем гнучкої розробки програмного забезпечення, (iii) визначення структури, яка надає рекомендації щодо використання переваг гнучкої розробки програмного забезпечення та моделей бізнес-процесів, вказуючи, як може працювати інтеграція (наприклад, налаштування команди та артефактів).

Ми також зазначили, що практична значущість результатів первинних досліджень у SLR не була широко підтверджена їхніми оцінками. З 37 відібраних

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		90

досліджень 27% були оцінені за твердженнями. Цей тип оцінки базується на експериментах, на які може вплинути упередженість, але вони також є основою для майбутніх реальних умов [40]. Тому ми розглядаємо їх як можливість для додаткових досліджень.

По-друге, список застосувань, визначений у Таблиці, надає основу для впровадження моделей бізнес-процесів у гнучких проектах розробки програмного забезпечення. Ми вважаємо, що цей список сам по собі може допомогти компаніям визначити застосування, які вони раніше не уявляли. Наприклад, компанія, можливо, вже досить довго використовувала моделі бізнес-процесів для доповнення історій користувачів (U10). Можливо, вони отримали різні переваги, роблячи це. Однак, можливо, вони не бачать інших застосувань моделей бізнес-процесів, які, залежно від середовища компанії, також могли б бути їм корисними. Визначення того, що робили інші, може надихнути їх. Це може, наприклад, призвести до створення загального уявлення про продукт за допомогою моделей процесів (U5) та пов'язати дії цієї моделі з історіями користувачів. Останнє може допомогти компанії візуалізувати вплив змін.

У зв'язку з цим списком застосувань ми представляємо додаткові результати: (i) фази проекту гнучкої розробки програмного забезпечення, де застосовуються ці використання та (ii) категоризація визначених застосувань відповідно до MN - Управління, TM - Команда, TH - Технологія, PR - Процес (див. Таблицю 8). Ці результати можуть допомогти практикам визначити стратегії, що вирішують проблеми в проектах гнучкої розробки програмного забезпечення. Незважаючи на позитивний відгук про Agile в індустрії програмного забезпечення, за останнє десятиліття дослідники та практики висвітлили кілька питань. Ці питання включають широкий спектр аспектів, пов'язаних з процесом його впровадження, якістю, безпекою, документацією, управлінням, відстежуваністю та комунікацією, і це лише деякі з них [39]. У цьому сенсі категоризація, зображена в Таблиці, може допомогти компаніям визначити, що використовувати та що вважається таким, що принесе більше

переваг. Прикладом такої ситуації може бути компанія, яка прагне вирішити - поширені проблеми процесів у гнучкій розробці (наприклад, підтримка проектної документації та специфікації вимог). У цьому випадку компанія може звернути свою увагу на використання моделей бізнес-процесів у Таблиці, де категорія «PR – Процес» має найбільшу підтримку. Залежно від конкретної - ситуації та використання категорії «PR – Процес», компанія може вирішити використовувати бізнес-моделі одним або кількома способами.

По-третє, корисність моделей процесів у Agile для практиків у цьому дослідженні полягає в тому, щоб підтримувати моделі якомога «простішими», наскільки це необхідно. Такі твердження підкреслюють необхідність досліджень інтеграції моделей бізнес-процесів у Agile з урахуванням практичних міркувань. Нехтування тим, як практики використовують моделі бізнес-процесів, може призвести до досліджень, які залишаться лише в академічній сфері.

Те саме спостереження стосується організацій, які створюють програмні інструменти для підтримки гнучкого процесу розробки програмного забезпечення з моделями бізнес-процесів. Вони повинні забезпечувати гнучкість для управління рівнем деталізації, необхідним для проектів.

3.4 Висновки по розділу

У третьому розділі було розглянуто реалізацію моделей та оцінку ефективності Agile-процесів. Проаналізовано результати моделювання та проведених експериментальних досліджень, що дозволило оцінити поведінку програмних процесів у реальних умовах. Виявлено основні обмеження Agile-підходів та запропоновано можливі шляхи їх подолання. Проведено оцінку ефективності побудованих моделей та інтерпретацію отриманих результатів, що підтверджує доцільність використання моделювання для оптимізації та вдосконалення Agile-процесів розробки програмного забезпечення.

					БР.ІІ – 50.00.00.000 ІЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено методи моделювання програмних процесів у Agile-середовищі, що є комплексним підходом до підвищення ефективності управління розробкою програмного забезпечення. У роботі проаналізовано сучасні Agile-методології, досліджено підходи до формалізації та моделювання процесів розробки, а також визначено ключові фактори, що впливають на продуктивність і якість програмних продуктів. Процес дослідження охопив аналіз вимог, побудову моделей, проведення експериментів і оцінку ефективності отриманих результатів, що дозволило сформулювати цілісне уявлення про застосування моделей у гнучкому середовищі.

На початковому етапі було проведено аналіз теоретичних основ Agile-підходів і визначено їх ключові принципи, зокрема ітеративність, адаптивність та орієнтацію на постійне вдосконалення. Визначення припущень, що лежать в основі гнучких процесів, дозволило окреслити обмеження традиційних підходів до планування та підкреслити необхідність використання методів моделювання для підвищення прозорості процесів. Це створило основу для подальшого дослідження методів аналізу та прогнозування у сфері розробки програмного забезпечення.

У другому розділі було розглянуто сучасні методи та інструменти моделювання, зокрема підходи системної динаміки, які дозволяють досліджувати поведінку складних систем у часі. Особливу увагу приділено використанню хмарних технологій та інструментів підтримки розробки, а також ролі людського фактора у формуванні ефективних команд. Аналіз показав, що поєднання технічних і організаційних аспектів є критично важливим для досягнення високих результатів у Agile-середовищі.

У процесі реалізації було проведено моделювання програмних процесів та виконано серію експериментів для оцінки ефективності запропонованих підходів. Отримані результати дозволили виявити закономірності впливу різних факторів на продуктивність команди, таких як розподіл ресурсів, швидкість

					БР.ІІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		94

виконання завдань і рівень взаємодії між учасниками процесу. Також було проаналізовано обмеження гнучких методологій, зокрема складність довгострокового планування та залежність від людського фактору, і запропоновано шляхи їх подолання.

Загалом результати роботи підтверджують, що використання методів моделювання у Agile-середовищі дозволяє підвищити ефективність управління проектами, покращити прогнозування результатів і оптимізувати використання ресурсів. Перевагами запропонованого підходу є можливість аналізу складних процесів, підвищення прозорості розробки та підтримка прийняття обґрунтованих управлінських рішень. Водночас існують певні обмеження, пов'язані зі складністю побудови моделей і необхідністю врахування великої кількості змінних.

У майбутньому результати дослідження можуть бути використані для розробки інтелектуальних систем підтримки прийняття рішень у сфері розробки програмного забезпечення, інтеграції з DevOps-практиками та автоматизації управління Agile-процесами. Це сприятиме подальшому підвищенню ефективності команд розробки та якості програмних продуктів.

					БР.ІІ – 50.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		95

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Agile Alliance. (2025). Agile Methodology Overview. agilealliance.org.
<https://www.agilealliance.org/agile101/>
2. Beck, K., et al. (2001). Manifesto for Agile Software Development. agilemanifesto.org. <https://agilemanifesto.org/>
3. Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. scrumguides.org. <https://scrumguides.org/>
4. Fowler, M. (2002). Patterns of Enterprise Application Architecture. martinfowler.com. <https://martinfowler.com/eaCatalog/>
5. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.
6. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach. McGraw-Hill.
7. Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Addison-Wesley.
8. Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development. Addison-Wesley.
9. Highsmith, J. (2009). Agile Project Management. Addison-Wesley.
10. Rubin, K. S. (2012). Essential Scrum. Addison-Wesley.
11. Cohn, M. (2005). Agile Estimating and Planning. Prentice Hall.
12. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice. Addison-Wesley.
13. Gamma, E., et al. (1994). Design Patterns. Pearson.
14. Freeman, E., & Robson, E. (2021). Head First Design Patterns. O'Reilly.
15. Martin, R. C. (2017). Clean Architecture. Pearson.
16. Humble, J., & Farley, D. (2010). Continuous Delivery. Addison-Wesley.
17. Kim, G., et al. (2016). The DevOps Handbook. IT Revolution.
18. Microsoft. (2025). Azure DevOps Documentation. learn.microsoft.com.
<https://learn.microsoft.com/en-us/azure/devops/>

					БР.ІІІ – 50.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		96

19. Google Cloud. (2025). DevOps and Agile Practices. cloud.google.com.
<https://cloud.google.com/devops>
20. AWS. (2025). Agile Development on AWS. aws.amazon.com.
<https://aws.amazon.com/agile/>
21. IBM. (2024). Agile Methodologies and Practices. ibm.com.
<https://www.ibm.com/topics/agile>
22. Atlassian. (2025). Agile Coach. atlassian.com.
<https://www.atlassian.com/agile>
23. Beck, K. (2004). Extreme Programming Explained. Addison-Wesley.
24. Cockburn, A. (2006). Agile Software Development. Addison-Wesley.
25. Sterman, J. D. (2000). Business Dynamics: Systems Thinking and Modeling. McGraw-Hill.
26. Forrester, J. W. (1961). Industrial Dynamics. MIT Press.
27. Richardson, G. P. (2011). Reflections on the Foundations of System Dynamics. System Dynamics Review.
28. Wolstenholme, E. (2003). Towards the Definition and Use of a Core Set of Archetypal Structures in System Dynamics.
29. Gartner. (2024). Agile Development Trends. gartner.com.
<https://www.gartner.com/en/information-technology>
30. McKinsey. (2024). Agile Transformation in Software Development. mckinsey.com. <https://www.mckinsey.com>
31. Deloitte. (2025). Agile and DevOps Insights. deloitte.com.
<https://www2.deloitte.com>
32. IEEE. (2023). Agile Software Development Research Papers. ieee.org.
<https://ieeexplore.ieee.org/>
33. ACM. (2024). Software Process Modeling Studies. acm.org.
<https://dl.acm.org/>
34. VersionOne. (2023). State of Agile Report. digital.ai.
<https://digital.ai/resource-center/analyst-reports/state-of-agile-report>
35. PMI. (2021). Agile Practice Guide. pmi.org. <https://www.pmi.org/>

36. OpenAI. (2025). AI in Software Development. openai.com.
<https://openai.com/research>
37. GitHub. (2025). GitHub Actions Documentation. docs.github.com.
<https://docs.github.com/actions>
38. GitLab. (2025). CI/CD Documentation. docs.gitlab.com.
<https://docs.gitlab.com/ee/ci/>
39. Stack Overflow. (2024). Developer Survey Results. stackoverflow.com.
<https://survey.stackoverflow.co/>
40. Fowler, M. (2021). Continuous Integration. martinfowler.com.
<https://martinfowler.com/articles/continuousIntegration.html>

					БР.ІІ – 50.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		98

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи бакалавра: " Методи моделювання програмних процесів у Agile-середовищі "

Обсяг пояснювальної записки: 95 аркушів

Дата закінчення дипломної роботи 10червня 2025р.

Підпис студента _____