Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Інтеграція традиційних методів розробки програмної архітектури в сучасні гнучкі технології розробки.

Виконав(ла): студент(ка) 6 курсу, групи СНмз-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Мариненко Л.Ю. (прізвище та ініціали)

Керівник (підпис) Марценюк В.П. (прізвище та ініціали)

Нормоконтроль (підпис) Дуда О.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Мариненко Леонід Юрійовтч
(прізвище, ім'я, по батькові)

1. Тема роботи **Інтеграція традиційних методів розробки програмної архітектури в сучасні гнучкі технології розробки**

Керівник роботи **д.т.н., проф. Марценюк В.П.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*24*» *листопада* 2023 року № *4/7-1098*

2. Термін подання студентом завершеної роботи 27 грудня 2023 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ 1.1 Задачі дослідження 1.2 Розробка та оцінка інструментів для емпіричного дослідження 1.3 Емпіричні дослідження 2 ВПРОВАДЖЕННЯ ТРАДИЦІЙНИХ МЕТОДІВ РОЗРОБКИ У AGILE-ПРОЄКТИ 2.1 Програмна архітектура в Agile-проєктах 2.2 Варіанти планування діяльності з розробки програмної архітектури 2.3 Роль архітектора ПЗ в Agile-проєктах 2.4 Інтеграція розробки архітектури у Agile-процеси розробки 3 РЕЗУЛЬТАТИ ЛІТЕРАТУРНОГО ОГЛЯДУ 3.1 Категорії практичних задач інтеграції розробки програмної архітектури у гнучкі проєкти 3.2 Кількісний аналіз результатів літературного огляду 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ ВИСНОВКИ СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ДОДАТКИ

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С, к.т.н., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.11.23-15.11.23	*Виконано*
2.	Підбір наукових джерел по темі роботи	16.11.23-20.11.23	*Виконано*
3.	Переклад та опрацювання наукових джерел	20.11.23-23.11.23	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.11.23-10.12.23	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	11.12.23-12.10.23	*Виконано*
6.	Оформлення другого розділу	12.12.23-13.12.23	*Виконано*
7.	Оформлення третього розділу	13.12.23-14.12.23	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	08.12.23-09.11.23	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	10.12.23-12.12.23	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	12.12.23-31.12.23	*Виконано*
11.	Нормоконтроль	14.12.23-15.12.23	*Виконано*
12.	Перевірка на плагіат	15.12.23	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.12.23	*Виконано*
14.	Захист кваліфікаційної роботи	28.12.2023	

Студент _____ Мариненко Л.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценюк В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Інтеграція традиційних методів розробки програмної архітектури в сучасні гнучкі технології розробки" // Кваліфікаційна робота освітнього рівня «Магістр» // Мариненко Леонід Юрійович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНмз-61 // Тернопіль, 2023 // с. – 56, рис. – 2 , табл. – 7, джерел – 37.

Ключові слова: Agile, моделі процесів розробки, розробка програмної архітектури, масштабування Agile-процесів

Механізми гнучких процесів розробки програмного забезпечення, що спрямовані на зниження витрат та оперативну реакцію на зміни на ринку, також виявились ефективними під час розробки складних програмних рішень. Недавні дослідження, які акцентують на розвитку гнучких процесів, вказують на можливість успішної співпраці та інтеграції елементів, які властиві як гнучким, так і традиційним підходам до розробки. У цьому контексті в рамках роботи виникає термін "традиціоналізація гнучких процесів".

Моделювання архітектури програмного забезпечення вважається однією з найбільш суттєвих справ у впровадженні елементів традиційних підходів у гнучкі процеси. Дослідження підтверджує, що просто нова архітектура в гнучких процесах недостатня для розробки складних програмних рішень, і вказує на необхідність визначених методів розробки архітектури в межах гнучких процесів. Результати дослідження демонструють, що включення явних архітектурних практик у гнучкі процеси розробки є корисним для успішного завершення проєктів.

ANNOTATION

“Integration of traditional methods for developing software architecture into modern Agile development technologies” // Master’s degree qualification paper // Marynenko Leonid Yuriyovych// Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНмз-61 // Ternopil, 2023 // p. – 56, fig. – 2, tables – 7, references – 37.

Key words: Agile methodologies, development process models, software architecture development, scaling Agile-processes

The mechanisms of agile processes, aimed at cost reduction and timely response to market dynamics, have also proven beneficial in developing complex software solutions. Recent research focused on expanding agile processes indicates a real possibility of coexistence and integration of additional elements from both agile and traditional development approaches. Within this work, this phenomenon is termed "agile process traditionalization."

Modeling software architecture stands out as one of the most sensitive issues when integrating traditional development elements into agile processes. Research confirms that a new architecture alone within agile processes is insufficient for developing complex software solutions and highlights the necessity for specific architectural development methods within agile processes. Study findings demonstrate that incorporating explicit architectural practices into agile development processes is beneficial for the successful completion of projects.

ЗМІСТ

ВСТУП 6
1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ 8
1.1 Задачі дослідження 10
1.2 Розробка та оцінка інструментів для емпіричного дослідження 12
1.3 Емпіричні дослідження 13
2 ВПРОВАДЖЕННЯ ТРАДИЦІЙНИХ МЕТОДІВ РОЗРОБКИ У AGILE-ПРОЄКТИ 15
2.1 Програмна архітектура в Agile-проєктах 15
2.2 Варіанти планування діяльності з розробки програмної архітектури. 19
2.3 Роль архітектора ПЗ в Agile-проєктах 22
2.4 Інтеграція розробки архітектури у Agile-процеси розробки 25
3 РЕЗУЛЬТАТИ ЛІТЕРАТУРНОГО ОГЛЯДУ 30
3.1 Категорії практичних задач інтеграції розробки програмної архітектури у гнучкі проєкти 30
3.2 Кількісний аналіз результатів літературного огляду 35
4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 41
4.1 Ергономічний аналіз умов праці. Система "людина-машина" 41
4.2 Оцінка хімічної обстановки та розрахунок аварії на підприємстві із зберіганням аміаку 45
ВИСНОВКИ 50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53
ДОДАТКИ

ВСТУП

Актуальність задачі.

Як форма організованого робочого процесу розробка програмного забезпечення не залишилася захищеною від феномену гнучкості. В історичній перспективі багато методів, які використовувалися з моменту розвитку найдавніших процесів розробки, охоплювали певні гнучкі елементи. Кінець минулого століття приніс деякі з найбільш часто використовуваних сьогодні процесів розробки, які повністю базуються на цінностях і принципах Agile. Ці процеси, тобто їх розширення, спрямоване на протистояння викликам, з якими вони стикаються сьогодні, є темою дослідження, представленого в цій роботі.

З метою подальшого уточнення теми дослідження, терміни, які найчастіше використовуються в цьому дослідженні, визначені далі. У рамках цієї роботи термін «традиційна розробка» означає процеси розробки, керовані планом, або так звані процеси важкої розробки, тоді як термін «гнучкі процеси» позначає ряд гнучких процесів розробки, таких як Scrum і XP, які повністю включають принципи та цінності, проголошені в Agile Manifesto 2001 року.

Незважаючи на те, що минуло більше, як два десятиліття з моменту публікації Agile Manifesto, популярність гнучких процесів розробки програмного забезпечення не зменшилася. Однак сьогодні вони стикаються з серйозними проблемами. Дедалі частіші зміни у бізнес-вимогах і зростаюча складність обставин, що стоять за цими змінами, ще більше ускладнюються розбіжністю фізичних і логічних аспектів бізнесу. Це вимагає більш швидкої адаптації інформаційних систем, які внаслідок цього стали більш неоднорідними та децентралізованими [5].

Дослідження в цій області також проводились і у ТНТУ. Зокрема, в [1] висвітлено питання впровадження традиційних методів в Agile-проєкти відповідно до теми цієї роботи. У [2], [3] розглядають питання проєктування програмної архітектури у сучасні гнучкі моделі управління проєктами. В [4] запропоновано методи вибору програмної архітектури в Agile-проектах.

Мета роботи.

Мета цієї роботи полягає в тому, щоб визначити, наскільки конкретні явні архітектурні практики підходять для включення в гнучкі процеси розробки. Для цього було проведено змішаний метод дослідження. Якісна складова дослідження призвела до ідентифікації явних архітектурних практик, придатних для застосування в процесах гнучкої розробки.

Їх значення визначали за допомогою кількісної складової, реалізованої у формі емпіричного дослідження.

Об’єкт дослідження: Процеси розробки програмного забезпечення.

Предмет дослідження: моделі життєвого циклу проєктів з Agile та традиційним методологіями.

Наукова новизна отриманих результатів. Запропоновано методи проєктування архітектури в якості традиційних для проєктів і команд в Agile проєктах.

Практичне значення отриманих результатів. У цій роботі проведено дослідження за літературними джерелами з метою розширення гнучких процесів за допомогою використання елементів, типових для традиційної розробки, які називаються «традиціоналізацією гнучких процесів», що дозволить інтегрувати традиційні методи управління проєктами в Agile-проєкти.

Апробація результатів та особистий внесок здобувача. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції "Інформаційні моделі системи та технології – 2023", та "Актуальні задачі сучасних технологій – 2023", які проводились у ТНТУ.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

Складність системи визначається трьома основними атрибутами: масштабом, відмінністю та зв'язністю. «Масштаб» відображає, скільки речей є в системі, «відмінність» визначається різноманітністю речей у системі, тоді як «зв'язність» відповідає тому, скільки різних зв'язків існує між речами [6].

В [7] представлено контекстну модель розробки програмно-інтенсивних систем, призначену для керівництва впровадженням та адаптацією практик гнучкої розробки програмного забезпечення. Модель виявилася ефективною у випадках, коли контекст проекту був суттєво віддалений від «Agile sweet spot», тобто контексту, з якого випливає Agile розробка, і в якому вона є найбільш успішною. Така модель описується, як «спільно розташована команда з менше, ніж 15 осіб, яка займається розробкою нових систем, що не мають критичного значення для безпеки, у досить нестабільному середовищі; архітектура системи визначена та стабільна, а правила управління зрозумілі».

Було визначено вісім факторів масштабування, які впливають на складність системи та середовища, в якому вона розроблена:

– Розмір команди – від менше ніж 10 розробників до сотні розробників.

– Географічний розподіл – від спільного розташування до глобального розподілу.

– Відповідність – від низького ризику до критичного/перевіреного.

– Організація та культура – від відкритих до вкорінених.

– Організація дистрибуції – від власних до сторонніх

– Управління – від неформального до формального..

– Складність програми – від простих одноплатформних до складних багатоплатформних.

– Корпоративна дисципліна – від фокусу на проєкті до фокусу на підприємстві.

У міру того, як область проекту переходить від «гнучкості», низька церемоніальність і висока ітераційність, як ключові характеристики гнучкості, більше не сприймаються як панацея. Крім того, існує очевидна тенденція поєднання взаємодоповнюючих елементів (донедавна вважалися конфліктними) гнучкої та традиційної розробки, що довело можливе їх співіснування та інтеграцію [8]. В роботі [9] подано порівняння моделей розробки Rational Unified Process, XP та Scrum. Після аналізу, спрямованого на пошук оптимального балансу між повторюваністю та церемоніальністю, автори запропонували комбіновану модель, яка включала б переваги та виключала обструктивні особливості RUP, XP та Scrum.

Архітектурні міркування є одними з найбільш делікатних питань при розгляді розширення гнучких процесів. Гнучкі процеси не пропонують типові, чіткі дії з розробки архітектури програмного забезпечення, такі як аналіз, синтез і оцінка, оскільки вважається, що вони призведуть до додаткових витрат, а не створять цінності для користувача [4]. Загалом, існує дві екстремальні архітектурні стратегії: Big Design Up Front (BDUF) і емерджентний дизайн. Прихильники гнучкої розробки вважають, що концепція метафори разом із методами рефакторингу є адекватними замінниками традиційного процесу розробки архітектури. Відповідно до них, архітектура розвивається поступово з кожною ітерацією, в результаті безперервних змін у вихідному коді (емерджентна архітектура) [10]. Однак, не заперечуючи, що гнучкі процеси пропонують організаціям ефективність, якість і гнучкість в управлінні змінами, деякі автори, наприклад [11] вважають, що чіткі архітектурні практики відіграють важливу роль у розробці складних програмних рішень. На їхню думку, рефакторинг, як архітектурна практика гнучких процесів розробки, може бути достатньо успішним лише за умови належного завершення високорівневого проектування архітектури програмного забезпечення. Це єдиний спосіб уникнути великої кількості рефакторингу, який спричинив би ескалацію витрат на розробку на наступних етапах, а також ерозію архітектури, що, можливо, поставило б під загрозу весь проект.

1.1 Задачі дослідження

Починаючи з припущення, що встановлення балансу між гнучкою та традиційною розробкою архітектури програмного забезпечення, або, точніше, між явними архітектурними практиками та гнучкістю процесу розробки, зрештою, сприятиме подоланню викликів, з якими стикаються гнучкі процеси при розробці дуже складних систем, ця стаття досліджує наступні питання дослідження:

1. Виконати дослідження літературних джерел щодо інтеграції традиційних методик у Agile-проєкти.

2. Ідентифікувати задачі, котрі виникають при проєктуванні архітектури складних інформаційних систем з використанням гнучких процесів розробки.

3. Які архітектурні практики рекомендовані для гнучких процесів розробки та розробки складних ІС.

Систематичний огляд літератури складається з трьох етапів: планування дослідження, проведення дослідження та звітування про дослідження. Етапи планування огляду: визначення потреби в огляді та розробка протоколу огляду. Етапи проведення огляду: ідентифікація дослідження, вибір первинних досліджень, оцінка якості дослідження, отримання та моніторинг даних, а також синтез даних.

Визначений протокол дослідження вимагав стратегії, на якій базувався б пошук первинного дослідницького матеріалу. Зокрема, стратегія передбачала:

– Визначення ключових слів для пошуку – Agile архітектура ПЗ, Agile методи та архітектура, Agility та архітектура.

– Вибір джерел для пошуку – як найпомітніші джерела наукових та фахових праць обрано бібліографічні бази даних Web of Science та SCOPUS.

– Визначення критеріїв для включення/виключення дослідницького матеріалу – дослідницькі та професійні статті, опубліковані в рецензованих журналах, а також матеріали конференцій/семінарів у період з 2000 року були визнані прийнятними, тоді як усі статті, які не пов’язували термін «гнучкість» із

процесами гнучкої розробки, статті які не базувалися на емпіричних дослідженнях або не мали валідного підходу/методу, а також роботи, які ґрунтувалися виключно на думках експертів, були виключені з аналізу.

Процес систематичного огляду літератури починався з початкового пошуку первинного матеріалу дослідження; Web of Science і SCOPUS використовувалися для ідентифікації статей із відповідними асоціаціями з визначеними ключовими словами. Результатом попереднього пошуку через Web of Science та SCOPUS став набір дослідницьких та професійних статей, опублікованих у рецензованих журналах та матеріалах конференцій/семінарів, відібраних відповідно до визначених ключових слів. Доступ до ідентифікованих документів здійснювався через такі електронні сервіси: IEEE Xplore, ACM Digital Library та ScienceDirect. Документи, які відповідали критеріям для включення, вже були знайдені в джерелах, перелічених у таблиці 1.1 [12].

Таблиця 1.1 – Результати пошуку по кожному електронному сервісу

Джерело	Кількість входжень із визначеними ключовими словами	Кількість документів, відібраних для подальшого аналізу	Кількість виключених документів
IEEE Xplore	700	45	650
Цифрова бібліотека ACM	237	11	230
ScienceDirect	46	14	36
Всього	983	70	916

Після попереднього аналізу, описаного вище, джерела, перелічені в таблиці 1.1, були досліджені відповідно до протоколу, визначеного для систематичного огляду літератури. Огляд загальної кількості звернень до кожного електронного сервісу наведено в таблиці 1.1. Результатом дослідження було 34 релевантних статті, 26 з яких були результатом первинного пошуку, а 8 – результатом вторинного пошуку. Вторинний пошук означає аналіз посилань, наведених у матеріалі первинного дослідження.

За допомогою визначеного протоколу дослідження фокус аналізу звузився до наукових статей, які безпосередньо стосуються питань дослідження. У систематичний огляд літератури не входили книги.

1.2 Розробка та оцінка інструментів для емпіричного дослідження

Емпіричне дослідження включало як якісну, так і кількісну складову. Відповідно було застосовано два інструменти дослідження: анкета для проведення інтерв'ю та анкета для проведення опитування [12].

Початковий набір запитань для інтерв’ю з експертами-практиками було визначено з метою відповіді на друге запитання дослідження шляхом вивчення стану практики. Метод інтерв’ю було обрано з наміром зібрати якомога більше інформації про контекст дослідження та практичні проблеми. На основі проаналізованих матеріалів дослідження були сформовані запитання.

Другий інструмент дослідження – анкета для проведення опитування – була створена з урахуванням попередньо завершеного систематичного огляду літератури та якісного аналізу даних інтерв’ю. Анкету розроблено в електронному вигляді за допомогою Google Forms. Вибраних експертів-практиків попросили відповісти на набір закритих запитань, поданих у формі шкал оцінювання і контрольних списків. Для відповіді на третє запитання дослідження (Емпіричні результати – Архітектурна практика) було проведено кількісний аналіз зібраних даних.

Оцінку інструментів дослідження (інтерв’ю та опитування) проводила група експертів у галузі гнучкої розробки та архітектури програмного забезпечення [12]. Кожне потенційне запитання оцінювалося за шкалою: 1 – незначний; 2 – дещо значний; 3 – значний; 4 - надзвичайно значний.

Після оцінювання було розраховано індекс валідності контенту для кожного питання, а також для всього інтерв’ю та опитування. Значення індексу валідності контенту для першої версії інтерв’ю становило 0,76, що вказувало на необхідність її доопрацювання відповідно до висновків експертів. Зміна

інтерв’ю передбачала виключення деяких питань з індексом валідності нижче 0,8, переформулювання окремих питань, об’єднання окремих питань в одне запитання тощо. Індекс змістовної валідності для всього опитування становив 0,83. Остаточний варіант анкети не містив запитань зі значенням індексу контентної валідності нижче 0,8 [12].

Остаточна версія інтерв’ю містила 40 відкритих запитань, розділених на 5 тематичних областей:

1) дані про респондента та контекст;

2) дані про моделі процесу розробки;

3) дані про виявлені проблеми та їх причини;

4) дані про архітектуру програмного забезпечення та

5) фактори контексту.

Остаточний варіант анкети включав 30 питань закритого типу.

1.3 Емпіричні дослідження

Цей розділ містить описи вибірки респондентів, засоби збору емпіричних даних та методи, які використовуються для кількісного та якісного аналізу даних.

Відбір респондентів. Природа досліджуваної проблеми вимагала цілеспрямованого відбору одиниць вибірки ($n \geq 20$), щоб дозволити застосування обох інструментів, розроблених для емпіричного дослідження. Цілеспрямована вибірка була необхідною, щоб уникнути включення осіб, які не мають необхідного типу та якості знань, умінь, досвіду, досвіду та інформації з проблемної сфери. Дослідження проводилося у відомих компаніях IT-сектору на «цілеспрямованій», однорідній вибірці з 20 експертів [12]. Одна й та сама група респондентів була використана як для інтерв’ю, так і для опитування.

Набір респондентів відбувався на основі визначеного списку критеріїв, яким мали відповідати потенційні учасники [12]:

– Знання та практичний досвід у розробці архітектури програмного забезпечення та складних систем з використанням гнучких процесів.
– Спроможність і бажання зробити внесок у дослідження.
– Підтвердження того, що вони приділять достатньо часу дослідженню.
– Хороші комунікативні навички.
– Академічна освіта в галузі інформаційних технологій.
– Більше 5 років професійного досвіду.

Збір та обробка емпіричних даних. Для забезпечення більшої точності та повноти даних інтерв'ю проводились "віч-на - віч" та записувалися (за згодою респондентів). Дані, зібрані під час інтерв'ю, транскрибувалися за допомогою текстового процесора та згодом піддавалися якісному аналізу за допомогою спеціального програмного забезпечення і тематичного аналізу контенту [13]. Результатом якісного аналізу стала ідентифікація категорій практичних архітектурних проблем, які є відповіддю на друге питання дослідження.

Як зазначалося раніше, опитування проводилося в електронному вигляді через Google Forms. Учасники отримали електронною поштою посилання на анкету разом із інструкціями щодо її заповнення. Учасникам була гарантована анонімність, конфіденційність даних, конфіденційність і добросовісне використання. Дані, отримані за допомогою анкети, були імпортовані з Google Sheets в MS Excel, де вони були підготовлені для кількісного аналізу в програмному комплексі SPSS.

Якщо бути точним, результати емпіричної вибірки з 20 експертів були більш статистично значущими для висновків, пов'язаних із загальною популяцією, завдяки вибірці завантажувального програмного забезпечення з 1000 реплікаціями. Результатом якісного аналізу став набір явних архітектурних дій, які респонденти-експерти оцінили як важливі для розробки складних систем і придатні для включення в гнучкі процеси розробки програмного забезпечення. Це дало відповідь на третє запитання дослідження.

2 ВПРОВАДЖЕННЯ ТРАДИЦІЙНИХ МЕТОДІВ РОЗРОБКИ У AGILE-ПРОЄКТИ

Твердження прихильників гнучкої розробки про те, що явні архітектурні практики непотрібні в гнучких процесах, є предметом дослідження більшості робіт, включених до систематичного огляду літератури. Проаналізована література свідчить про те, що емерджентна архітектура може бути життєздатною альтернативою звичайним підходам до розробки архітектури програмного забезпечення, але лише в деяких архітектурних сферах, у той час як вона абсолютно неадекватна для інших.

Наприклад, у [14] стверджується, що емерджентна архітектура є хорошою практикою для детального проектування, але вона не охоплює набір важливих архітектурних дій, які повинні відповісти на те, чи рішення виконує ті функції, що повинно. Ці заходи передбачають спілкування із зацікавленими сторонами, спрямоване на розуміння їхніх потреб, визначення вимог і подолання протиріч і конфліктів між визначеними вимогами. Усунення цієї явної архітектурної діяльності збільшує реальний ризик того, що неправильні рішення, прийняті на етапі проектування, залишаться непоміченими, поки не стане надто пізно.

2.1 Програмна архітектура в Agile-проєктах

Результати огляду літератури також показують, що нефункціональним вимогам не приділяється достатньо уваги в процесі проектування. Це часто пояснюється тим, що реалізація нефункціональних вимог здійснюється потім, шляхом змін у вихідному коді під час супроводу системи, оскільки він триває довше та має більший бюджет. Проте у випадку великомасштабної та складної розробки систем звичайний гнучкий процес має бути розширений, щоб включити такі чіткі архітектурні дії, пов’язані з нефункціональними вимогами:

– тестова розробка з зосередження на нефункціональних вимогах;

– прототипування з акцентом на нефункціональних вимогах і моніторинг технічного стану з акцентом на нефункціональних вимогах.

В [15] ввели поняття архітектурно відповідальної особи, яка відповідає за визначення та аналіз нефункціональних вимог. Автори мали намір покращити процес виявлення, аналізу та управління архітектурно значущими вимогами шляхом введення концепції персон з різних областей.

Автори [16] запропонували метод, який розширює Scrum трьома явними архітектурними діями, зосередженими на нефункціональних вимогах:

1) аналіз і управління нефункціональними системними вимогами (після аналізу функціональних вимог);

2) відображення функціональних і нефункціональних вимог за допомогою матриці асоціацій зв'язків, яка представляє їх кореляцію, щоб забезпечити відстеження та завершення як продукту, так і беклогу спринту;

3) перевірка виконання нефункціональних вимог після кожного спринту.

Якщо процес верифікації показує, що раніше ідентифікована нефункціональна вимога не була виконана, навіть якщо всі функціональні можливості, з якими вона пов'язана, були реалізовані, ці функції повинні бути переглянуті, або має бути розроблена нова стратегія для виконання даної нефункціональної вимоги. сформульовано.

В [17] також підкреслили необхідність чіткої ідентифікації нефункціональних системних вимог, щоб підтримувати виявлення залежностей між функціональними та нефункціональними вимогами та архітектурними елементами на кожній ітерації. Як функціональні, так і нефункціональні вимоги повинні бути пріоритетними, щоб можна було визначити належний графік для кожного випуску. Відповідальністю архітектора є надання нефункціональних системних вимог як цінності користувачам, а також впровадження їх у тісній співпраці з програмістами.

Явна архітектурна діяльність із визначення архітектурних структур не входить до емерджентної архітектури [14], як у випадку з передбаченням майбутніх системних змін, що є діяльністю, критичною для прийняття рішень

щодо часу реалізації конкретної архітектурної діяльності на основі вартості. Архітектурне планування передбачає архітектурні міркування, які виходять за рамки поточної ітерації, спрямовані на передбачення майбутніх вимог, які має підтримувати архітектурне рішення [14, 17].

Планування, обмежене однією ітерацією, призводить до дегенерації дизайну та втрати гнучкості, що може перешкоджати гнучкості всього проекту. Основною причиною цього є те, що функціональні вимоги не можна аналізувати та розробляти повністю окремо, оскільки вони взаємозалежні. Функціональні вимоги з високою бізнес-цінністю для користувача і, відповідно, високим пріоритетом, часто залежать від вимог з нижчою бізнес-цінністю, які необхідно реалізувати першими.

Окрім аналізу взаємозалежностей між функціональними вимогами, також необхідно аналізувати залежності між функціональними можливостями з одного боку та нефункціональними вимогами та архітектурними елементами системи з іншого. В іншому випадку підвищується ризик неадекватності реалізованих проектних рішень, що може призвести до збільшення технічного браку в майбутньому. Поширення технічного браку з часом викликає проблеми, які не можуть бути вирішені тільки шляхом модифікації вихідного коду, а вимагають радикальних змін в архітектурі.

На додаток до вибору функціональних можливостей, які повинні бути реалізовані в рамках ітерації, запропонована концепція розширює планування релізу, включаючи ідентифікацію архітектурних елементів, які необхідно розробити для підтримки функціональних можливостей і майбутніх змін. Розміщення всіх проектних дій у поточній ітерації є надзвичайно небезпечною стратегією, особливо в проектах розробки програмного забезпечення у великих бізнес-організаціях, що характеризуються великою кількістю різноманітних додатків (давно розроблюваних і нових), різними технологіями та великою кількістю команд [18].

В [19] автори ввели термін «епічні архітектури» для позначення розширення масштабу архітектурного планування за межі однієї ітерації в Scrum.

Епічна архітектура – це архітектура, розроблена для узгодженої групи функціональних вимог. Метою епічної архітектури є визначення загальних елементів. Вона розроблена приблизно для 8 запланованих спринтів. Розпізнавання та реалізація їх схожості в поточному спринті зменшує загальні зусилля, необхідні для впровадження, одночасно збільшуючи одноманітність функціональних вимог, які необхідно реалізувати в наступних спринтах. Завдання реалізації для спринту виводяться на основі визначених архітектурних вимог, з яких складаються архітектурні юзер-сторіз.

Згідно з дослідженнями, встановлення балансу між масштабним архітектурним плануванням і новою архітектурою все ще є складною проблемою. Аналіз документів, присвячених цьому питанню, показує загальну позицію, що попередній дизайн повинен бути адекватним з точки зору такого балансу, що означає, що слід уникати пастки стратегії BDUF (Big Design up Front), типової для традиційної розробки (рис. 2.1).

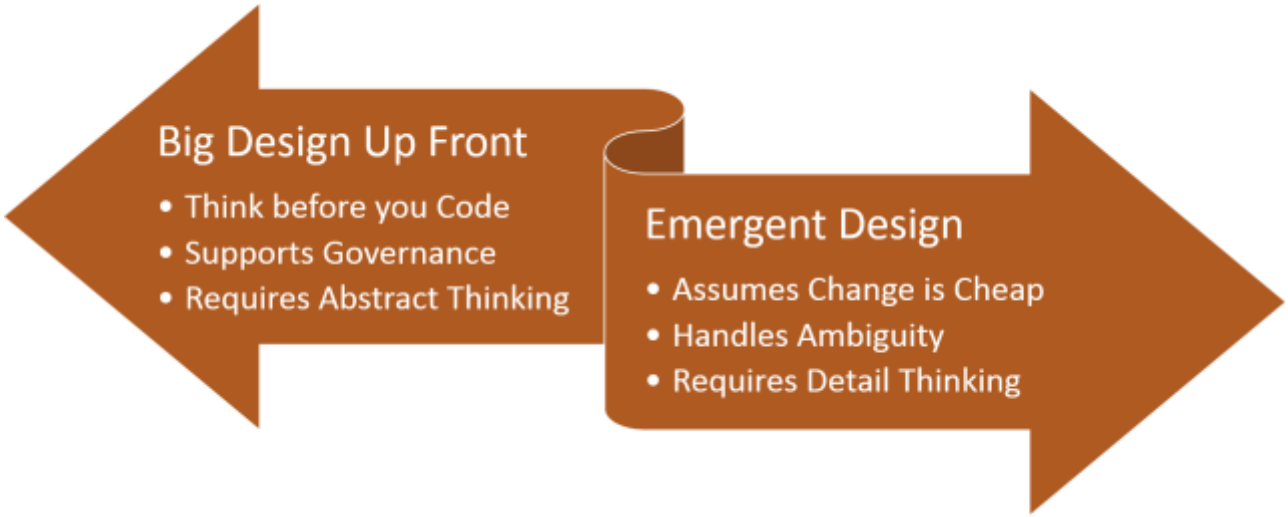

Рисунок 2.1 – Принципи традиційною та емерджентної розробки програмної архітектури

Наступний пункт містить огляд поглядів на це питання, які зустрічаються в проаналізованій літературі.

2.2 Варіанти планування діяльності з розробки програмної архітектури

Автор [14] вказує на те, що оцінка відповідного обсягу попереднього аналізу та проектування залежить від досвіду, навичок, знань архітекторів програмного забезпечення, а також ефективної комунікації із зацікавленими сторонами, тоді як автори [20] додали ще два чинники – розуміння обраного архітектурного рішення та використання попередньо визначеної архітектури (з точки зору існуючих шаблонів, архітектури, рекомендованої постачальниками, або інструментів, що використовуються для автоматизації процесу). Автори [17] також стверджують, що планування архітектури не має бути надто масштабним, а скоріше «достатнім». Вони запропонували підхід, заснований на трьох концепціях:

– аналіз залежності;
– аналіз реальних варіантів і
– управління дефектами.

Аналіз залежностей передбачає перевірку та керування залежностями між функціональними вимогами, залежностями між функціональними та нефункціональними вимогами, а також залежностями між вимогами та архітектурними елементами. Метою аналізу залежностей є сприяння своєчасній розробці архітектурних елементів, які можуть підтримувати впровадження необхідних функцій. Це вимагає планування архітектури, яке виходить за рамки однієї ітерації, тобто передбачення та аналіз майбутніх потреб.

Аналіз обох видів вимог можна представити за допомогою однієї таблиці. Після цього виконується аналіз DSM – Dependencies Structure Matrix (матриця структури залежностей), щоб виявити залежності між усіма складовими підсистемами, які представляють певні функціональні вимоги (наприклад, обмін даними між підсистемою продажів і закупівель), а потім аналіз DMM – Domain Mapping Matrices (матриці відображення домену), щоб визначити їх залежність

від певних архітектурних елементів (такі, як компоненти інтерфейсу користувача, процедури доступу до даних, безпека тощо).

Важливість процесу аналізу архітектури програмного забезпечення в гнучких процесах також підкреслили автори роботи [21], які дійшли висновку, що найефективнішою технікою є аналіз залежностей, але на рівні коду. Фокус запропонованої методики зосереджений на виявленні статичних залежностей від вихідного коду, щоб порівняти реалізовану архітектуру із запланованою. Виявлені залежності використовуються як індикатор зв'язків між цими двома рівнями архітектури, що разом із стандартними гнучкими методами (безперервне тестування, безперервний аналіз коду, безперервна інтеграція коду, безперервний рефакторинг і парне програмування) сприяє додатковому безперервному контролю якості.

Після завершення аналізу залежностей в [17] вважають оптимальний вибір необхідних архітектурних елементів ключовим для реалізації масштабованості релізу. Для цього автори запропонували використовувати теорію реальних опціонів – модель фінансового аналізу, яка використовується в корпоративних фінансах для оцінки економічної ефективності конкретних бізнес-рішень.

Аналіз реальних варіантів може бути ефективно використаний у плануванні релізів, для розподілу архітектурних елементів для конкретних релізів. Аналіз реальних варіантів може оптимізувати інвестиції в конкретні архітектурні рішення на основі результатів аналізу залежностей і аналізу витрат і вигод відповідного архітектурного рішення. Остаточне рішення також має бути обґрунтованим з точки зору зменшення ризику, пов'язаного з майбутніми невизначеностями. Мета полягає в тому, щоб знайти відповідне та економічно ефективне рішення сьогодні, не ставлячи під загрозу можливість розробки більш повного рішення завтра.

Теорія реальних варіантів також використовується в [22], але для вирішення проблеми знаходження правильного часу для прийняття архітектурних рішень, оскільки автори вважають, що архітектурні рішення в гнучких процесах приймаються або занадто рано, або занадто пізно. Вони

запропонували структуру, яка повинна керувати командами у визначенні найбільш підходящого моменту для прийняття конкретних архітектурних рішень. Структура передбачає використання електронної таблиці, у якій фази розробки перераховані в стовпцях, а архітектурні питання складають рядки. Пропонований фреймворк спрямований на те, щоб зберегти конструкцію в певних межах, щоб уникнути пастки BDUF.

Автори [23] також досліджували вдосконалення процесу прийняття архітектурних рішень, починаючи з припущення, що архітектурні рішення в гнучких проектах приймаються програмістами точно вчасно, тоді як архітектор відіграє консультативну роль у цьому процесі.

Автори представили фреймворк 3A (agile, architecture, axes) на основі трьох осей, які необхідно враховувати, щоб створити єдиний процес прийняття архітектурних рішень, який підійде для конкретного проекту. Перша вісь (who – хто) містить ролі з потенційними обов'язками в процесі прийняття архітектурних рішень (команда розробників, архітектор додатків, архітектор домену, архітектор підприємства, менеджмент).

Друга вісь (how – як) містить засоби документування архітектурних рішень для комунікації із зацікавленими сторонами (пряме спілкування, нотатки зустрічей, спеціальна документація – вікі, неофіційна документація на основі шаблонів і офіційна документація на основі шаблонів).

Третя вісь (when – коли) містить різні періоди від моменту прийняття архітектурного рішення до моменту отримання відгуку про його обґрунтованість (більше 6 місяців, 1-6 місяців, менше 1 місяця).

Автор [24] запропонував набір евристик для вирішення проблеми балансування розробки функціональності та архітектури програмного забезпечення. Реалізація запропонованої евристики вимагає комунікація між різними ролями в проекті, спрямованого на вивчення восьми запропонованих вимірів: семантики, обсягу, життєвого циклу, ролей, документації, методу та вартості. Відповідаючи на ключові питання в цих областях, учасники розробляють загальну ментальну модель щодо застосування певних

архітектурних практик. Після цього вони можуть визначити процес управління, а також технічний процес, який керує архітектурною діяльністю в рамках гнучкого процесу (так звана «модель блискавки»).

Автори [25] запропонували модель C3A, яка інтерпретує концепцію Agile архітектури як синтез так званої еталонної архітектури, яка ілюструє бачення в технічних і функціональних термінах (це результат планування), і архітектури реалізації, який включає частини еталонної архітектури в майбутній реліз (розвиток функціональності). Метод включає процес оцінки еталонної архітектури та еволюції архітектури впровадження, який використовується для аналізу розриву між тим, що було заплановано, і тим, що було досягнуто в рамках випуску. Ключові кроки цього методу включають: слухати та спостерігати, спостерігати, розмірковувати, покращувати, ретельно вивчати та запускати.

2.3 Роль архітектора ПЗ в Agile-проєктах

Декілька авторів досліджували роль архітекторів програмного забезпечення в досягненні балансу між попередньою та новою архітектурою в гнучких процесах розробки.

Автор роботи [26] вважає, що зміна традиційної ролі архітекторів є необхідною для збалансованого інвестування у функціональні/нефункціональні вимоги. Він підкреслює, що архітектори несуть відповідальність за загальну якість системи і що їхній вибір адекватних дизайнерських рішень впливає на баланс між реалізацією функціональних і нефункціональних системних вимог. Передбачається, що архітектори надають послуги як програмістам, так і клієнтам, маючи кілька ролей у взаємодії з ними. Вони повинні надавати цінність клієнтам через реалізацію нефункціональних системних вимог, а також надавати постійну підтримку програмістам протягом усього процесу впровадження.

Автори роботи [27] також заявили, що важливо включити архітектора програмного забезпечення в увесь процес гнучкої розробки, але наголошують на

необхідності тісної співпраці архітектора з командою розробників із постійним обміном ідеями протягом усього проекту.

Роль архітектора програмного забезпечення є незамінною в розробці та постачанні великих, складних систем, які зазвичай вимагають розробки та інтеграції кількох систем і координації сотень людей. Архітектор програмного забезпечення є єдиною людиною, яка приймає рішення щодо життєво важливих аспектів системи, оскільки навіть старші програмісти часто не можуть це зробити.

Роль архітектора програмного забезпечення полягає в тому, щоб на самому початку проекту розглянути проблему, яка вирішується, з різних точок зору, оскільки кожна бізнес-проблема унікальна і вимагає різних підходів до виявлення її відмінних аспектів. Після того, як архітектор програмного забезпечення завершить концептуальний опис системи, Agile–команда може вирішити, як її протестувати (наприклад, динамічне виконання, статичне тестування або моделювання). За допомогою раннього інструментування та валідації цього процесу архітектор повинен перетворити своє бачення на загальне (всієї команди проекту), і контролювати розробку, а також реагувати у разі неочікуваних проблем або необхідних змін.

В [28] вказано, що архітектори програмного забезпечення повинні бути тими, хто ліквідує розрив між гнучким процесом розробки та методами розробки архітектури програмного забезпечення. Щоб розробити гнучку архітектуру, яка врівноважує як традиційний, так і гнучкий підхід, архітектори програмного забезпечення повинні мати виняткове розуміння гнучкого процесу, а також здатність створювати баланс між бізнесом і архітектурними пріоритетами. Це можна зробити з допомогою гібридної моделі Scrum, XP і послідовного управління проектами для розробки гнучкої архітектури. Автор виступає за використання архітектурних функцій і навичок (спілкування, нефункціональні вимоги, вибір відповідного програмного та апаратного забезпечення, шаблони проектування) протягом чотирьох етапів процесу:

– попереднє планування,

– розкадрування,

– спринт,

– робоче програмне забезпечення.

Автори [29] зазначають, що необхідно встановити баланс початкових архітектурних заходів, щоб уникнути загрози концепції гнучкості у великомасштабних і складних проектах. Вони пропонують, щоб кожен проект починався з попереднього аналізу ризиків, щоб ідентифікувати та ізолювати складні області, а також ідентифікувати елементи програмного забезпечення, інфраструктуру та архітектуру даних. Початок проекту передбачає опис задачі на концептуальному рівні разом із пропозиціями вирішення її у заданій сфері. Іноді проблема/рішення досягаються шляхом аналізу існуючих систем або шляхом вибору комерційного рішення для певних областей. Якомога раніше архітектор повинен також вирішити, як будуть перевірені найбільш ризиковані аспекти кожної виявленої проблеми, визначити архітектурно важливі вимоги та, якщо необхідно, розробити прототипи.

Для середніх і великих проектів розробки було запропоновано модифікувати та розширити етапи розробки процесу XP:

– планування проекту,

– аналіз та управління ризиками,

– проектування та розробка,

– тестування.

Автори [30] запропонували гібридну модель для великих і складних гнучких проектів, засновану на інтеграції процесу XP з методами розробки архітектури програмного забезпечення, розробленими Інститутом розробки програмного забезпечення Університету Карнегі-Меллона (метод аналізу компромісів архітектури, воркшоп атрибутів якості, метод проектування керований атрибутами, метод аналізу витрат і вигод, активні аудити для проміжного проектування). Ці методи можуть підвищити цінність гнучких

процесів, оскільки вони підкреслюють нефункціональні вимоги та їх значення в проектуванні архітектури.

2.4 Інтеграція розробки архітектури у Agile-процеси розробки

Сьогодні децентралізація, неоднорідність і потреба у сумісності є одними з найважливіших викликів, з якими стикається бізнес. У розробці таких систем зазвичай бере участь багато членів команди, у той час як системи часто потрібно масштабувати до найвищого рівня продуктивності та безпеки. Вони часто є критично важливими, і розуміється, що вони не повинні зазнати невдачі. Все це вимагає потужної архітектурної підтримки системи та відповідної документації. З іншого боку, користувачі очікують, що ці програмні рішення будуть адаптованими до змін у бізнес-середовищі, що вимагає застосування принципів гнучкої розробки.

В роботі [31] автори визначили три чинники, які необхідно враховувати при встановленні балансу між архітектурою та застосуванням принципів гнучкої розробки в проектах розробки складних систем: розмір системи, критичність і мінливість вимог. Вони запропонували підхід, заснований на кількісній оцінці витрат і ризиків (за допомогою моделі COCOMO II і концепції фактора вирішення ризиків (RESL)) інвестицій в архітектуру. На основі результатів дослідження автори запропонували гібридну (гнучку/керовану планом) структуру процесу для розробки таких складних систем – модель інкрементального зобов’язання (ICM – Incremental Commitment Model). Фреймворк являє собою синтез концепцій із існуючих моделей процесів: концепції ранньої верифікації та валідації з моделі V, концепції паралелізму в паралельній моделі, концепції полегшених процесів з Lean та інших гнучких моделей, концепцій, керованих ризиками зі спіральної моделі, а також фази та опорні точки з раціонального уніфікованого процесу (RUP).

Автори в роботі [32] запропонували спосіб встановлення балансу між початковою та емерджентною архітектурою шляхом застосування концепції

Lean для управління потоком цінностей протягом усього процесу розробки. Відповідно до концепції Lean, усі відходи від архітектурної діяльності можна розділити на три тісно пов’язані категорії: перевиробництво, затримка та брак. Результати їхнього дослідження показали, що поступова розробка архітектури спричиняє збільшенню витрат, пов’язаних із вищезгаданими затратами. Результатом дослідження стали вказівки щодо вдосконалення процесу розробки за рахунок більш ефективного управління часом і утримання затрат у виробництві. Автори вважають, що стратегія розробки архітектури багатьма малими кроками може зменшити витрати на затримки, пов’язані з часом, необхідним для завершення всього проекту архітектури.

Однак рефакторинг архітектури може бути набагато дорожчим, оскільки може знадобитися значні зміни. Розробка з меншою кількістю більш великих кроків зменшує витрати, пов’язані з рефакторингом, але збільшує витрати на затримку через більшу кількість архітектурних робіт. Витрати на затримку виникають через очікування або затримку процесу реалізації, тоді як витрати на рефакторинг виникають через архітектурні недоліки або затрати, спричинені перевиробництвом. Вони представляють дві крайні стратегії планування ітерації. Для того, щоб розробка йшла проміжним шляхом з точки зору витрат, автори пропонують використовувати концепцію візуалізації інвестицій в архітектуру в межах кожного кроку, щоб продемонструвати вплив затрат на архітектуру (через перевиробництво, затримку або дефект) на всю архітектуру. З цією метою вони пропонують визначити архітектурно важливі вимоги, а також прийнятне тестування, щоб забезпечити видимість архітектурних завдань на дошці Канбан.

Автори в [33] зазначають, що візуалізація архітектури має вирішальне значення для встановлення балансу між передовою та новою архітектурою, особливо у випадку нестабільного середовища. Вони вважають, що традиційна техніка рефакторингу коду не є адекватним рішенням для проблем, що виникають через складність коду та побічних ефектів, викликаних швидкою розробкою, адаптацією до змін у бізнесі та оновленням системи. Натомість вони пропонують підхід вищого рівня – рефакторинг на основі архітектури; процес,

керований архітектурним планом, який є результатом визначених залежностей на структурному рівні. Рефакторинг створюється прототипом перед застосуванням до вихідного коду. Він складається з п’яти кроків:

– визначення проблеми,

– візуалізація поточної архітектури,

– моделювання бажаної архітектури з точки зору поточних елементів,

– консолідація та переупакування кодової бази,

– автоматизація керування архітектурою шляхом постійної інтеграції.

Емпіричні дані показують, що аналіз на основі архітектури може підвищити продуктивність розробки програмного забезпечення та знизити витрати на обслуговування системи.

Рефакторинг архітектури можна включити в гнучку розробку як обов’язковий процес, оскільки це може покращити раннє виявлення та усунення неадекватних або неоптимальних проектних рішень і забезпечити високу якість архітектури. Цю дію потрібно виконувати принаймні один раз за ітерацію. Пропонується підхід до систематичного виконання процесу рефакторингу архітектури, який включає наступні ключові етапи:

– оцінка архітектури (виявлення проблем дизайну);

– пріоритезація (визначення послідовності вирішення архітектурних питань, виходячи з їх значущості);

– вибір шаблону (якщо він існує) для кожної виявленої проблеми або традиційний редизайн;

– забезпечення якості шляхом оцінювання або тестування, щоб гарантувати, що рефакторинг не спричиняє випадкових змін семантики.

Архітектурні проблеми, які слід вирішити за допомогою цього процесу, включають: нечіткі ролі об’єктів, складність архітектурних рішень, надмірну централізацію, асиметричну структуру чи поведінку тощо. Якщо проблеми залишаться невирішеними своєчасно, відбудеться ерозія дизайну, і рефакторинг більше не виконуватиметься для вирішення архітектурних питань, що

залишилися. У такому випадку рішеннями, що залишилися для відновлення архітектури, є реінжиніринг або реконструкція.

Автори роботи [34] пропонують забезпечити видимість архітектурних структур через застосування фреймворку, який пов'язує архітектурні рішення з вихідним кодом. Процес починається з опису архітектури у файлі XML, який згодом використовується для генерації коду, що реалізує описану архітектуру, слугуючи скелетом програми. Фреймворк дозволяє кільком командам працювати над одним кодом із мінімальними конфліктами, якщо залежностями у вихідному коді керують ефективно. Це дає змогу програмістам мати справу з функціональністю компонентів, тоді як реалізована структура керує компонентами, інтерфейсами та їхніми залежностями.

Результати огляду літератури свідчать про те, що значна частина проблем виникає через один додатковий істотний конфлікт: вимога мінімалізму в гнучких процесах і потреба в добре задокументованій архітектурі в складних системах. Дані з виробництва ПЗ показують, що в більшості випадків архітектурна документація або надлишкова, або зовсім відсутня. Невідповідна документація призводить до недостатності архітектурної інформації та знань, поганого розуміння архітектури та порушення комунікації, що призводить до хаосу та провалу проекту.

Надмірна документація спричиняє марну витрату часу та ресурсів, а також відхід від суті. Саме архітектор програмного забезпечення, який виконує роль постачальника послуг, відповідає за підтримку центральної позиції між недостатньою та надмірною документацією інструкцій щодо розробки. І хоча є опубліковані емпіричні результати, які свідчать про те, що гнучкі команди розробників знаходять архітектурні артефакти корисними для полегшення спілкування між членами команди розробників на останніх етапах проектування, а також для документування та оцінки рішення, але традиційна практика документування, загальний план архітектури програмного забезпечення зазвичай використовуються лише для концептуального опису архітектури.

У літературі є кілька пропозицій щодо подолання описаної проблеми документування. Наприклад, робота [35] пропонує вихід у документуванні архітектурних рішень Автори запропонували шаблон для документування архітектури програмного забезпечення, який відповідає філософії Agile та документації Lean. Вони пропонують застосувати концепцію управління знаннями про архітектуру, яку вони модифікували та інтегрували з процесом Scrum. Цей підхід передбачає розробку архітектурної бази даних і застосування методу оцінки архітектури на основі прийняття рішень.

Результати теоретичних досліджень свідчать про досить багато проблем архітектури, якими займаються різні автори.

Таблиця 2.1 – Архітектурні проблеми, визначені в літературі

№	Актуальна архітектурна проблема
1	Нефункціональні вимоги
2	Передбачення майбутніх вимог і уявлення про архітектуру поза поточним випуском
3	Баланс між початковою та емерджентною архітектурою
4	Роль архітектора програмного забезпечення
5	Наочність архітектурних завдань
6	Документування архітектури

Список цих основних категорій проблем подано в табл. 2.1.

ЗРЕЗУЛЬТАТИ ЛІТЕРАТУРНОГО ОГЛЯДУ

3.1 Категорії практичних задач інтеграції розробки програмної архітектури у гнучкі проєкти

Практичні архітектурні питання були визначені за результатами архітектурного огляду на основі інтерв’ю з експертами з гнучкої розробки. Подібні теми були об’єднані в концепції, а подібні концепції були розділені на 8 категорій практичних архітектурних питань (табл. 3.1). Визначені категорії практичних архітектурних питань вказують на необхідність застосування певних явних архітектурних практик у розробці архітектури програмного забезпечення в гнучких процесах. Виявлені категорії є відповіддю на друге питання дослідження.

Порівняння архітектурних питань в існуючій літературі (таблиця 2.1) і архітектурних питань, виявлених після аналізу емпіричних даних, зібраних під час інтерв'ю (таблиця 3.1), показує багато збігів. Іншими словами, проблеми, які розглядаються дослідниками, відповідають проблемам, виявленим на практиці. Наступний текст містить інтерпретацію результатів, наведених у таблиці 3.1.

Емпіричні результати показують, що вимоги є важливим фактором, коли справа доходить до вибору архітектурної стратегії, оскільки їхні риси впливають на стратегічну орієнтацію по відношенню до двох крайнощів – BDUF і емерджент – і, отже, є джерелом численних архітектурних проблем.

Більшість респондентів заявили, що вони працюють у нестабільному середовищі, оскільки клієнти часто не мають чіткого бачення того, що їм потрібно, що є додатковою причиною затримок на етапі впровадження. Це означає, що вимоги, з якими працює архітектор програмного забезпечення на початку проекту, є неповними. Якість вимог має значний вплив на кількість часу, який архітектор програмного забезпечення повинен присвятити попередньому аналізу архітектури, щоб визначити обсяг основної частини програмного забезпечення.

Таблиця 3.1 – Категорії задач інтеграції традиційних методів розробки архітектури в Agile-розробку

Категорія	Концепція
Функціональні вимоги	1. Неповні вимоги 2. Непостійні вимоги
Нефункціональні вимоги	1. Неадекватне врахування та визначення нефункціональних вимог 2. Неналежний моніторинг виконання нефункціональних вимог 3. Не тестування нефункціональних вимог 4. Нехтування рефакторингом, який має покращити якість дизайну 5. Несвоєчасне вирішення питань, пов'язаних з нефункціональними вимогами 6. Технічна заборгованість
Бачення архітектури	1. Відсутність стратегічного архітектурного планування 2. Не враховуючи майбутні системні вимоги 3. Неадекватний розподіл часу на дослідження та аналіз архітектурних вимог 4. Ігнорування певних аспектів, важливих для розвитку архітектури 5. Нехтування вибором адекватного архітектурного рішення для швидшої реалізації функціональності
Техніко-технологічні аспекти	1. Невивчення можливостей і обмежень поточних технологій, фреймворків і бібліотек сторонніх розробників 2. Незбалансоване застосування традиційних і нових технологій 3. Недостатнє знання використовуваної технології 4. Неадекватний вибір технології та основи реалізації розв'язуваної проблеми
Бізнес аналіз і розуміння проблеми	1. Недостатньо часу для процесу аналізу бізнесу 2. Погане розуміння проблеми 3. Відсутність предметних знань
Архітектурна оцінка	1. Нехтування перевіркою виконання нефункціональних вимог 2. Нечасте створення прототипів, що має запобігти поганому дизайну 3. Неформальний процес перевірки архітектури 4. Нехтування показниками та тестами 5. Рідке використання обмеженого за часом підтвердження концепції

Продовження таблиці 3.1

Категорія	Концепція
Роль архітектора програмного забезпечення	1. Координуюча роль архітекторів у робочому проектуванні 2. Деякі архітектурні рішення викликають вузькі місця
Команда	1. Недосвідчені члени команди 2. Обмежена пропозиція на ринку праці
Документування архітектури	1. Неадекватне управління архітектурними знаннями 2. Архітектурні рішення та їх причини здебільшого залишаються недокументованими

Відсутність прихильності до визначення архітектурно значущих вимог призводить до збільшення загальних витрат на проект, а також його тривалості. Нестабільність вимог є другою за частотою проблемою, з якою стикаються гнучкі архітектори програмного забезпечення та гнучкі команди. Нерідкі випадки, коли клієнти повністю змінюють своє уявлення про те, чого вони очікують від програмного рішення на етапі впровадження. Найкращий спосіб пом’якшити цей ризик – докласти більше зусиль на початковій стадії проекту, перед налаштуванням архітектурного рішення для основної частини системи. Відповідно до цього гнучкі команди повинні визначити набір архітектурно важливі вимоги до основної частини системи на початку проекту, залишаючи ідентифікацію інших вимог та ітераційне вдосконалення розробленого архітектурного скелета для фази впровадження.

Практикуючі спеціалісти також зазначили проблемну сферу визначення нефункціональних вимог через те, що зацікавлені сторони здебільшого не знають про них і роблять занадто великий акцент на реалізації функціональних вимог у вимогах. Немає систематичного моніторингу реалізації нефункціональних вимог, на відміну від функціональних вимог, які відстежуються через Беклог і Беклог Продукту. Тестування нефункціональних вимог також не є послідовною та обов’язковою практикою, як у випадку з функціональними вимогами. Ось чому на практиці нехтують своєчасним рефакторингом, який має покращити якість дизайну; натомість дизайн «виправляється» для кожної команди, це

виправлення потрібне для подолання гострої проблеми та забезпечення реалізації наступних функціональних вимог. Такий підхід передбачає несвоєчасне вирішення питань, пов’язаних з нефункціональними вимогами та постійною наявністю технічного браку. Запропоноване вирішення цієї проблеми полягає у створенні та постійному оновленні єдиного пріоритетного списку функціональних і нефункціональних вимог і відображенні всіх завдань на дошці Kanban.

Друга група архітектурних питань пов'язана з баченням архітектури. Коли справа доходить до вибору архітектурного рішення, найпоширенішим підходом в галузі є «когнітивне дослідження» архітектора, а також мозковий штурм з іншими архітекторами та членами команди розробників. Вибираючи рішення, архітектори здебільшого покладаються на особистий досвід і знання, а також на експертизу команди розробників. Через часові обмеження вибране архітектурне рішення часто не є найкращим для вирішуваної задачі. У більшості випадків відповідальність за вибір архітектурного рішення несе клієнт.

Практики помітили, що їм потрібно більше часу для дослідження та аналізу архітектурних вимог і підтвердження архітектурних рішень. Крім того, вони усвідомлюють той факт, що, намагаючись розпочати реалізацію якомога швидше, вони не враховують усі важливі архітектурні аспекти на початку проекту (наприклад, розгортання), а також майбутні вимоги до системи. Це передбачає низку наслідків і більші витрати на останніх фазах розробки. Насправді немає балансу між стратегічним і тактичним плануванням архітектури, тому є ризик зробити неправильний поворот у розробці. Це збільшує ризик архітектурної ерозії. На додаток до того факту, що зміна основних архітектурних рішень на останніх фазах розробки є дорогою, сценарій, за якого архітектуру неможливо виправити рефакторингом, також залишається можливим.

Практичні результати виявили категорію проблем архітектури, пов’язаних з техніко-технологічними аспектами розвитку архітектури. Яскраве зображення цих проблем ілюструється тим, що необхідно приймати архітектурні рішення

щодо того, де і як буде виконуватися програмний продукт. Тоді технологічні аспекти є ключовим фактором у розробці архітектури, і тому необхідно приділити достатньо часу для вивчення можливостей і обмежень поточних технологій, фреймворків і сторонніх бібліотек на початку проекту. Це важливо для адекватного вибору технології та основи реалізації з огляду на проблему, що вирішується. Однаково важливо, щоб члени команди, особливо архітектор програмного забезпечення, були знайомі з використовуваною технологією: останні тенденції в архітектурних параметрах, технологічні інновації та компоненти сторонніх виробників, які можна використовувати в розробці. Крім того, респонденти зазначили, що існує тенденція використання нових технологій, без чітких аргументів на користь такого вибору. Пояснення базуються виключно на аргументі, що «нове» краще за старе. Такий спосіб мислення є неправильним, і його слід замінити збалансованим застосуванням традиційних і нових технологій з огляду на їх переваги у вирішенні проблеми.

Однією з серйозних проблем респонденти визнали недостатній час для аналізу бізнесу та вивчення проблеми, що призводить до неадекватних архітектурно значущих вимог. Час, витрачений на аналіз, прямо пропорційний рівню предметних знань про проблему, з якою мають справу члени команди, перш за все, архітектор ПЗ.

Практики вважають, що для успіху архітектури програмного забезпечення критично важливо, щоб архітектор досліджував проблему з різних точок зору на самому початку проекту, оскільки кожна бізнес-проблема є різною та має унікальні архітектурні аспекти. Найважливішим для вирішення бізнес-проблеми є досягнення розуміння проблеми через розуміння того, як працює цільова організаці, тобто її бізнес-процеси, оскільки вони є основою для визначення архітектурно значущих вимог. Отримання розуміння бізнес-процесів залежить від чіткої архітектурної діяльності, яка передбачає обговорення з ключовими зацікавленими сторонами на групових зустрічах або індивідуальних інтерв’ю. Респонденти відзначили важливість участі архітектора програмного забезпечення в зустрічах з власниками продукту, оскільки інакше вони ніколи не

отримають всю необхідну інформацію з документації. Хоча власники продуктів часто (але не обов’язково) є технічним персоналом, вони не мають того ж рівня знань і досвіду, що й архітектори програмного забезпечення, і тому не можуть передати 100% своїх вимог.

Архітектурна оцінка передбачає перевірку архітектури щодо архітектурно значущих вимог. Результати свідчать про те, що цей сегмент архітектурних питань здебільшого базується на спеціальному підході. Крім стандартних гнучких практик (перегляд коду, інтеграція коду, регресійне тестування, статичний і динамічний аналіз коду тощо), респонденти усвідомлюють важливість кількох традиційних практик (прототипування, обмежене в часі підтвердження концепції, формальний огляд експертами в архітектурі), але вони також заявили, що зазвичай не вистачає часу на ці техніки і що вони застосовують їх лише тоді, коли це необхідно.

Практики також усвідомлюють той факт, що метрики та тести є найкращим засобом перевірки архітектури, тобто виконання нефункціональних вимог, і що вони прагнуть їх використовувати. Однак вони також заявили, що все ще переглядають архітектуру ad hoc, без визначеного процесу. Перевірка виконання нефункціональних вимог найчастіше відводиться на етап обслуговування, щоб прискорити доставку цінності користувачам через розвиток функціональних можливостей або через бюджетні обмеження.

Формальна роль архітектора програмного забезпечення не є стандартною для гнучких процесів. Гнучкі команди є багатофункціональними, і всі члени команди поділяють відповідальність за архітектуру. Проте гнучкі команди, які беруть участь у дослідженні, зазвичай виконують офіційну роль архітектора програмного забезпечення, яку виконує досвідчений програміст. Як досвідчений програміст, архітектор програмного забезпечення бере участь у групі розробників на початку проекту, щоб разом із програмістами налаштувати основну частину програмного забезпечення. Як сходяться на думці кількох авторів, роль архітекторів програмного забезпечення в гнучкій команді має суттєво відрізнятися від традиційної, оскільки архітектори повинні бути

постійно задіяні протягом усього процесу розробки. Їхня роль повинна включати координацію протягом усієї розробки програмного рішення. Проте респонденти зазначили, що єдина роль архітектора – узгодження робочого проекту.

Щоб досягти координуючої ролі архітектора програмного забезпечення протягом усього процесу розробки, практики пропонують стратегію, спрямовану на «…підвищення обізнаності, довіри, навичок і знань у проблемній області та технології…» серед усіх членів команди. Покращення рівня технічних знань членів команди можна досягти найефективніше, включивши їх до обговорення архітектури, коли вона вперше розробляється на початку проекту, а також під час планування ітерації. Дотримуючись цього підходу, команди можуть уникнути того , щоб архітектори були вузьким місцем у разі радикальної зміни дизайну через архітектурне рішення.

Таким чином, інші члени команди, які інакше не мали б навіть базових архітектурних знань або повної картини рішення, можуть допомогти прискорити впровадження архітектурного рішення. Agile-команди усвідомлюють, що відповідальність за архітектуру має лежати на всій команді, а не лише на архітекторах. Однак такий підхід вимагає кваліфікованих людей з високим рівнем технічних знань, що є основною проблемою, з якою гнучкі команди стикаються на практиці. Швидкий розвиток IT-індустрії з одного боку, і обмежена пропозиція на ринку праці, з іншого, призводять до того, що гнучкі команди складаються здебільшого з недосвідчених людей. Причини цієї проблеми, виявлені в дослідженні, пов’язані з постійним припливом нових працівників, а також з тим, що середній інженер в команді займає молодший рівень.

Документація – це дуже важливе архітектурне питання. Традиційна розробка надмірно акцентує увагу на цій діяльності, тоді як гнучкі процеси майже замінюють її ідеєю вихідного коду як остаточної документації, відповідно до проголошеної гнучкої цінності «робочого програмного забезпечення над вичерпною документацією». Однак, коли йдеться про розробку складних програмних рішень, якісний вихідний код не є достатньою практикою

документування; важливо помірковано включати певні практики документування. Архітектурна документація здебільшого написана у вікі і містить описи архітектурно значущих функціональних і нефункціональних вимог, а також рішення щодо технологічного стеку. Він також містить базові архітектурні моделі, намальовані вручну у вигляді блок-схем. Формальні моделі рідко використовуються на практиці, оскільки вимагають багато часу та зусиль для постійного перегляду, що зменшує гнучкість. Проте проблема в тому, що на більшість архітектурних рішень та їх причини немає документації. Як наслідок такої відсутності стратегії управління архітектурними знаннями, більшість архітектурних знань залишається «в пастці» у свідомості людей, і тому їх неможливо повторно використати. Гнучкі команди визнають цю проблему, але досі не мають її рішення.

3.2 Кількісний аналіз результатів літературного огляду

Зібрані дані були піддані кількісному аналізу з метою визначення значення явних архітектурних практик для процесів гнучкої розробки. Кількісний аналіз проводився для кожної архітектурної практики, описаної вище, і приклад аналізу для конкретної гнучкої практики – визначення ключових зацікавлених сторін системи – наведено в наступному тексті.

Частоти оцінок значущості спостережуваної архітектурної практики представлені в таблиці 3.2 як у цифрах, так і у відсотках. Рядки таблиці 3.2 відповідають змінним чотирирівневої шкали оцінювання з опитувальника (1 – незначно; 2 – дещо суттєво; 3 – суттєво; 4 – надзвичайно суттєво).

Таблиця 3.2 – Архітектурні практики: визначення ключових зацікавлених сторін системи

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent	
						Bias	Std. Error
Valid	1.00	2	10.0	10.0	10.0	.0	6.6
	2.00	3	15.0	15.0	25.0	.1	8.4
	3.00	12	60.0	60.0	85.0	.2	11.1
	4.00	3	15.0	15.0	100.0	-.3	7.8
	Total	20	100.0	100.0		.0	.0

Таблиця 3.3 містить верхній і нижній відсотки оцінок, розраховані з використанням 95% довірчого інтервалу, обчисленого за допомогою початкової повторної вибірки з 1000 реплікаціями. Наприклад, відсоток балу 3, розрахований із 95% довірчим інтервалом, становить від 40 до 80.

Таблиця 3.3 – Архітектурні практики: визначення ключових зацікавлених сторін системи

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	1.00	.0	25.0
	2.00	.0	35.0
	3.00	40.0	80.0
	4.00	.0	30.0
	Total	100.0	100.0

На основі результатів, отриманих таким чином, для кожної архітектурної практики були складені зведені таблиці даних, які показують їх значення для процесів гнучкої розробки (таблиця 3.4). Значущість явних архітектурних практик вимірюється як частка респондентів, які оцінили їх як важливі.

Таблиця 3.4 – Значущість архітектурних практик в Agile-проєктах

Архітектурна практика	Значущість арх. практики
Формування відповідної команди та вибір архітектора програмного забезпечення з урахуванням проблеми, що вирішується	Дуже значущий: 0,95
Розуміння бізнес-проблеми	Дуже значущий: 0,95
Огляд коду	Дуже значущий: 0,95
Активні дискусії зі стейкхолдерами з метою аналізу та розуміння бізнесу	Високозначуща: 0,9
Виявлення архітектурно значущих вимог	Високозначуща: 0,9
Аналіз ризиків, спрямований на виявлення та виділення областей складності	Високозначуща: 0,9
Вивчення технології, придатної для впровадження	Високозначуща: 0,9
Ідентифікація та визначення базових структур (модулів) для ядра системи, а також їх зв’язків (передбачувана архітектура)	Високозначуща: 0,9
Тестування продуктивності системи та інших критичних нефункціональних вимог	Високозначуща: 0,9
Обсяг проекту	Дуже значущий: 0,85
Аналіз залежностей між функціональними вимогами та архітектурними елементами під час планування випуску	Дуже значущий: 0,85
Безперервна підтримка архітектора в усіх ключових питаннях проектування	Дуже значущий: 0,85
Створення спільного пріоритетного списку функціональних і нефункціональних вимог	Високозначуща: 0,8
Визначення базової архітектури даних	Високозначуща: 0,8
Експертиза та розробка моделі розгортання	Високозначуща: 0,8
Перевірка критичних архітектурних вимог через прототипування	Високозначуща: 0,8
Специфікація інтеграційних тестів	Високозначуща: 0,8
Визначення ключових зацікавлених сторін системи	Значущий: 0,75
Огляд офіційної архітектури	Значущий: 0,75
Специфікація тестового випадку	Значущий: 0,75
Планування випуску зі стратегією, зосередженою на дослідженні застарілих систем, залежності від продуктів інших партнерів/третіх сторін і зворотній сумісності даних	Значущий: 0,7
Специфікація на приймальні випробування	Значущий: 0,7
Розробка/оцінка QA тестів	Значущий: 0,7
Розробка інструкцій з написання коду та інших інструкцій для проектування системи	Незначний: 0,6
Розробка документації верхнього рівня	Незначний: 0,55
Управління залежностями із зовнішніми системами, з якими система взаємодіє протягом випуску	Незначний: 0,5
Технічне управління боргом із зосередженням на нефункціональних вимогах у кожній ітерації	Незначний: 0,45
Визначення детального дизайну кожного модуля	Незначний: 0,4
Детальна проектна документація	Незначний: 0,4
Вивчення та вдосконалення робочого проекту	Незначний: 0,4

Значущість явних архітектурних практик вимірюється як частка респондентів, які оцінили їх як важливі.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Ергономічний аналіз умов праці. Система "людина-машина"

На певному етапі свого розвитку для задоволення своїх зростаючих матеріальних і духовних потреб людина починає створювати штучні знаряддя праці – машини. Одержавши у своє розпорядження величезні запаси енергії, нову техніку й технології, вона змінила своє життя, але разом з тим постала перед складним завданням – забезпечити ефективне, стійке та безпечне керування цією технікою.

При вирішенні завдань пов’язаних з поліпшенням умов праці, необхідне детальне вивчення системи „людина-машина” (СЛМ). СЛМ – це складна багатофункціональна система, яка включає в себе людський і технічний фактори (рис. 4.1) і має такі складові:

– машина – усе те, що штучно створено руками людини для задоволення своїх потреб (технічні пристрої, інформаційне забезпечення);

– людина – людина-оператор, при взаємодії з машиною виконує деякі функції для досягнення поставленого завдання;

– навколишнє середовище – визначається такими параметрами, як освітленість, шум, випромінювання, температура, вологість тощо;

– робоче місце – окреслюється положенням оператора при виконанні своїх обов’язків;

– органи керування (ОК) – за допомогою їх людина керує об'єктами;

– засоби відображення інформації (ЗВІ) – завдяки їм людина слідкує за станом машини (виробничого процесу).

Одним із важливих завдань СЛМ є розподіл функцій між людиною і машиною, який повинен ураховувати їх можливості. Однак загальне рішення складно отримати, оскільки кожна система характеризується своїми особливостями. На основі порівняння можливостей людини і машини в системах керування можна запропонувати наведений далі варіант розподілу функцій.

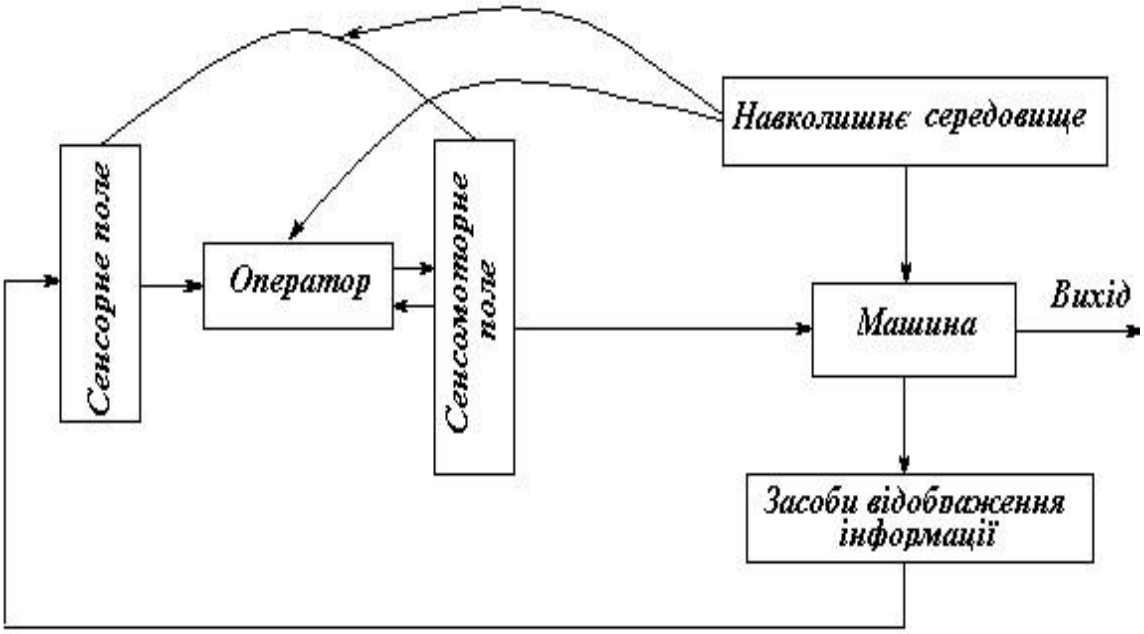

Рисунок 4.1 – Структурна схема системи „людина-машина”

Людина виконує такі функції:

– індуктивно мислить, тобто приймає рішення на базі неповної інформації, узагальнення різних факторів, доповнюючи інформацію з власного досвіду;

– розпізнає ситуацію в цілому за її окремими характеристиками, а також за не повною інформацією про неї;

– вирішує завдання, стосовно яких відсутні правила;

– вибирає шляхи вирішення завдань в умовах, що швидко змінюються. Машині доцільно передати такі функції:

– виконання громіздких математичних розрахунків та вибір відомих варіантів розв’язання;

– збереження великої кількості інформації;

– здійснення одноманітних операцій за відомим алгоритмом;

– виконання швидких дій у відповідь на певну команду.

Ці рекомендації мають узагальнений характер і у кожному конкретному випадку визначальними є експеримент з моделювання конкретної системи та умов функціонування, а також застосування певних принципів. Тому дуже

важливо оцінити умови, в яких буде працювати людина, щоб, виходячи з них, розподілити обов’язки.

Процеси приймання, переробки інформації та прийняття рішень і виконання оператором керуючих дій поєднанні в цілісну діяльність, яка полягає у гарантуванні функціонування СЛМ. Для ефективного забезпечення роботи машини людина-оператор повинна зручно себе відчувати. Таке можливо при виконанні певних вимог, які ставляться до машини і навколишнього середовища, а саме: до розміщення засобів відображення інформації та засобів керування, робочого місця, а також до освітлення, клімату, шуму, вібрації, що можуть впливати як на фізичний стан людини, так і на протікання технологічного процесу.

Згідно з ДСТУ EN 17025 рівень якості продукції визначається сукупністю операцій, які включають вибір номенклатури показників якості оцінюваної продукції, визначення їх величин і зіставлення з базовими. На основі ергономічної оцінки виробничого устаткування можна скласти портрет промислового підприємства, тобто описати організацію процесу виготовлення продукції, характеристику тієї частини основних фондів виробництва, яка безпосередньо впливає на якість продукції і залежить від людського фактору.

Класифікація показників ергономічної устаткування:

– антропометричні (висота, ширина, глибина пульта, висота розміщення стільниці, розміщення ЗВІ та ОК, характеристики крісла людини-оператора; досяжність ОК; показники відповідності ОК формі і розмірам частин тіла людини тощо);

– біомеханічні (зусилля, величина, напрямок переміщення ОК, частота використання ОК);

– психофізіологічні (характеристики відповідності техніки зоровому і слуховому аналізаторам людини);

– психологічні (показники відповідності техніки можливостям людини стосовно прийому, обробки інформації та прийняття рішень).

Оцінка починається із складання плану проведення досліджень. Основні етапи таких досліджень зазвичай виконуються за такою схемою:

а) ознайомлення з призначенням, метою системи, завданнями та основними вимогами до неї;

б) побудова структурної схеми, що відтворює зв’язки окремих підсистем, потоки інформації і хід регулювання. При цьому окремо виділяються ланцюги, де задіяна людина-оператор з позначенням прямих і зворотних зв’язків у СЛМ (інтенсивність зв’язків і їх відносна важливість);

в) оцінка середовища, в якому система функціонує, і його вплив на досліджувану систему СЛМ;

г) опис функцій системи і її підсистем для всіх режимів роботи (включаючи малоймовірні аварійні ситуації). При визначенні функцій системи слід зазначити, які операції найважливіші і що являє собою динамічна структура системи, тобто які зрушення виникають в окремих підсистемах при керуванні, дії перешкод і т. ін.

д) детальна ергономічна оцінка робочого місця;

е) оцінка засобів відображення інформації та органів керування;

ж) розгляд функцій операторів для нормального режиму роботи й окремо для екстремальних ситуацій.

На підставі всіх вищезазначених дій формується висновок про надійність і ефективність системи й даються рекомендації щодо модернізації або вдосконалення окремих підсистем, вузлів або всього приладу.

При вирішенні завдань, пов’язаних з пристосуванням умов праці до людини, необхідне детальне вивчення системи „людина-машина”, що являє собою складну багатофункціональну систему і включає: людину, машину, навколишнє середовище, органи керування, засоби відображення інформації, робоче місце. Основне завдання – забезпечити максимальну продуктивність при мінімальних затратах енергії. Для цього потрібно оцінити можливості людини, з’ясувати фактори, які погіршують працездатність, та забезпечити відповідне розміщення засобів відображення інформації, органів керування на робочому

місці, з метою мінімізації їх впливу як на фізичний стан людини, так і на протікання технологічного процесу.

4.2 Оцінка хімічної обстановки та розрахунок аварії на підприємстві із зберіганням аміаку

У комплексі заходів захисту населення та об'єктів господарювання від наслідків надзвичайних ситуацій (НС) значне місце займає оцінка хімічної обстановки. Оцінка обстановки в загальному плані включає визначення:

- масштабу та характеру НС;
- заходів, які необхідні для захисту населення;
- доцільних дій при ліквідації НС;
- оптимального режиму роботи об'єкта господарювання в умовах НС.

Необхідність оцінки хімічної обстановки випливає з небезпеки ураження людей сильно діючими отруйними речовинами (СДОР), що потребує швидкого втручання, враховуючи її вплив на організацію рятувальних та невідкладних аварійно-відновлюваних робіт, а також на виробничу діяльність об'єкту господарювання в умовах хімічного зараження.

Масштаби та ступінь хімічного зараження місцевості залежать від кількості СДОР, їх складу, відстані від місця аварії, метеоумов.

Оцінка хімічної обстановки включає визначення:

- розмірів зон хімічного зараження;
- часу підходу зараженого повітря до певного рубежу (об’єкту);
- часу вражаючої дії СДОР;
- вибору найбільш доцільних варіантів дій, за яких виключається ураження людей.

Основні вихідні дані при оцінці хімічної обстановки:

- тип СДОР;
- кількість СДОР;
- метеоумови та топографічні умови місцевості;

– ступінь захищеності людей, укриття, техніки та майна.

Задача оцінки хімічної обстановки.

На об'єкті зруйнована необвалована ємність з 5-ма тонами аміаку. 50% із 600 працюючих забезпечені протигазами. На відстані 2 км розміщений населений пункт з 1000 мешканцями, на 40% забезпечених протигазами. Місцевість відкрита. Інверсія, швидкість вітру 1 м/с.

Визначити:

1. Глибину, ширину площі розливу і ЗХЗ.
2. Час підходу зараженого повітря до населеного пункту.
3. Можливі страти серед працюючих і мешканців, їх структуру.

Розв'язок.

Визначення глибини зони хімічного забруднення.

При оптимальних умовах для розповсюдження хмари забрудненого повітря – місцевість відкрита, ємність необвалована, швидкість вітру 1 м/с, ступінь вертикальної стійкості повітря – інверсія, глибина розповсюдження хмари, забрудненої НХР з поправочним коефіцієнтом на інверсію 5.

Для аміаку глибина розповсюдження хмари становить 0,7 км. З врахування поправки для інверсії $Г = Г_1 \times 5 = 0,7 \times 5 = 3,5$ км.

Поправочних коефіцієнтів на обваловану ємність, швидкість вітру та закриту місцевість не використовуємо відповідно до умови задачі.

Визначення ширини зони хімічного забруднення.

Визначимо ширину зони хімічного забруднення за формулою:

$$Ш = К_ш \times Г \text{ (км)},$$

де

$К_ш$ –коефіцієнт ширини ЗХЗ, який при інверсії =0,03, ізотермії – 0,15, конвекції – 0,8

Тому, $Ш = 0,03 \times 3,5$ км $= 0,105$ км.

Площа зони можливого хімічного забруднення

Площу зони хімічного забруднення (ЗХЗ) конкретно до умов забруднення визначаємо за формулою:

$$S_{змхз}=0,5\times Г \times Ш\ (км^2)$$

Отже, $S = 0,5 \times 3,5 \times 0,105 = 0,18\ км^2$

Площа розливу НХР.

У випадку зруйнування не обвалованої ємності визначається площа розливу НХР за формулою

$$S_р= G : (d \times h)\ (м^2),$$

де

G – кількість розлитої НХР (т) ;

h – висота шару розливу НХР – 0,05 м ;

d – густина НХР (т/м^2), для аміаку – 0,68 т/м^3.

Тому $S_р= 5 / (0,68 \times 0,05) = 147,1\ м^2$.

Визначення часу підходу хмари забрудненого повітря.

Час підходу хмари забрудненого повітря до населеного пункту визначаємо за формулою :

$$t_{підх}=R : (60 \times W)\ (хв),$$

де

R – відстань від джерела забруднення до заданого об’єкту (м) ;

W – середня швидкість переміщення хмари забрудненого повітря (м/с), яку знаходимо за формулою W=(1,5…2)V,

де

V – швидкість вітру в приземному шарі повітря м/с; 1,5 при R<10км (в нас відстань до населеного пункту становить 2 км за умовою задачі);

Отже, $t_{підх} = 2000 /(60 \times 1,5 \times 1) = 22,2$ хв.

Отже, час, за який необхідно провести оповіщення та евакуацію населення із зони хімічного забруднення, повинен бути меншим, ніж 22,2 хв.

Визначення тривалості уражаючої дії аміаку.

Час уражаючої дії НХР залежить від часу її випаровування із забрудненої поверхні, площі розливу та швидкості вітру.

Тривалість уражаючої дії первинної хмари складає 20-30 хв., тривалість уражаючої дії вторинної хмари визначається часом повного випаровування ОР із забрудненої ділянки.

Тривалість уражаючої дії аміаку знаходимо за формулою:

$$T = t_в \times К_в \text{ (год)},$$

де $t_в$ – час випаровування НХР (табл.2)

$К_в$ – поправочний коефіцієнт на час випаровування НХР при швидкості вітру понад 1 м/с (примітка до табл.2). Згідно умови задачі $К_в = 1$.

Згідно таблиці 2 і примітки до неї знаходимо час випаровування ($t_в$) аміаку, який при швидкості вітру 1 м/с = 1,2 год. для необвалованої ємності.

Тому $T = t_в = 1{,}2$ год.

Визначення можливих втрат в осередку хімічного ураження.

Можливі втрати в ОХУ залежать від умов розташування людей, захищеності їх засобами індивідуального захисту (забезпеченості їх протигазами) і визначаються за формулою :

$$В = NB_1 \text{ (чол)},$$

де N – кількість людей (чол);

$В_1$ – можливі втрати людей в осередку ураження (%) в залежності від умов їх розташування і забезпеченості протигазами (табл.3).

У нашій задачі для відкритої місцевості при забезпеченні протигазами 50% $В_1$=50%

Тому В = 600 чол × 50% = 300 чол.

Структуру втрат населення визначаємо згідно примітки до табл.3

Легко уражені $В_{лу} = 300 \times 25\% = 75$ чол.

Уражені середнього і тяжкого ступеня $В_{ст} = 300 \times 40\% = 120$ чол.

Смертельно уражені $В_{су} = 300 \times 35\% = 105$ чол.

Результати оцінки хімічної обстановки занесено у таблицю 4.1.

Таблиця 4.1 – Результати розрахунків

Джерела забруднення	Вид НХР	Кількість НХР, м	Глибина ЗХЗ, км	Ширина ЗХЗ, км	Площа ЗХЗ, ЗМХЗ, км^2	Площа зони розливу, м ^{2}	Час підходу забрудненого повітря, хв	Тривалість уражаючої дії НХР, год.	Можливі втрати, % (чол.)
Зруйнована необвалована ємність	Аміак	5	3,5	0,105	0,18	147,1	22,2	1,2	300

Для жителів населеного пункту при забезпеченні протигазами 40% B_1=58%.

Тому В = 1000 чол × 58% = 580 чол.

Структуру втрат населення визначаємо згідно примітки до табл.3

Легко уражені $B_{лу}$ = 580 × 25% = 145 чол.

Уражені середнього і тяжкого ступеня $B_{ст}$ = 580 × 40% = 232 чол.

Смертельно уражені $B_{су}$ = 300 × 35% = 203 чол.

Висновок: Оскільки населений пункт потрапляє в зону хімічного ураження, тривалість уражаючої дії аміаку становить 1,2 год., час підходу хмари забрудненого повітря 22,2 хв., тому необхідно терміново провести оповіщення мешканців населеного пункту про загрозу хімічного ураження та необхідність негайно одягнути протигази і евакуюватись в напрямку, перпендикулярному до руху хмари забрудненого повітря.

ВИСНОВКИ

Традиційні процеси розробки, які колись успішно відповідали бізнес-вимогам і потребам у розробці складних програмних рішень, більше не можуть адекватно відповідати на нові бізнес-виклики, з якими стикаються сучасні організації. Це причина, чому гнучкі процеси набувають популярності при розробці складних систем. Найбільшою проблемою при розробці складних систем з використанням гнучких процесів є розробка міцної архітектури, зберігаючи при цьому гнучкість процесу розробки. Емерджентної архітектури виявилося недостатньо для розробки сильної архітектури складних систем. Отже, гнучкі процеси необхідно розширити шляхом включення значних явних архітектурних практик.

На відміну від традиційної розробки, де архітектурна діяльність зосереджена на початкових етапах проекту, у гнучких процесах архітектурні питання розглядаються протягом усього життєвого циклу розробки. Однак результати дослідження показують, що практики Agile оцінили явні архітектурні практики з початкових етапів традиційного розвитку (планування та визначення обсягу) як найбільш важливі.

Незважаючи на те, що це твердження протистоїть Agile Manifesto, який виступає за «реагування на зміни замість дотримання плану», а також принцип XP «YAGNI» (You Ain't Gonna Need It), очевидно, що традиційні архітектурні практики часто застосовуються в гнучких процесах розробки. Ефективне управління такими проектами розробки програмного забезпечення вимагає системного підходу до встановлення балансу між традиційними та гнучкими архітектурними стратегіями. Крім того, очевидно, що гнучкі архітектори високо цінують такі дії, як тестування та перевірка коду.

Результати також свідчать про те, що гнучкі практики не мають бажання документувати і детальний дизайн, який підтверджує твердження про те, що вони намагаються знайти оптимальний баланс між гнучким і традиційним підходами. Що стосується детального планування дизайну, то гнучкі практики

не вважають це важливою архітектурною діяльністю, а розглядають це як відповідальність програмістів. Респонденти не вважають вихідний код єдиною необхідною формою документації в проектах розробки складних систем.

Діяльність із документування зведена до мінімуму та здебільшого виконується за допомогою вікі-сайтів без формального шаблону чи структури. Найчастіше документуються архітектурно значущі вимоги, рішення про технологічний стек і архітектурні моделі у вигляді блок-схеми. Існує очевидна проблема серед практиків, пов’язана з відсутністю документації щодо причин та обґрунтування більшості архітектурних рішень.

Незважаючи на те, що гнучкі процеси формально не визнають роль архітектора програмного забезпечення, дослідження показують, що така роль існує в гнучких командах. Однак він відрізняється від традиційного. Принципова відмінність полягає в безперервній участі архітекторів програмного забезпечення протягом усього процесу розробки.

Архітектор програмного забезпечення повинен пов’язувати гнучкий процес з методами розробки архітектури програмного забезпечення. Таким чином, вони повинні мати чітке розуміння гнучкого процесу, встановити баланс між пріоритетами бізнесу та архітектури, щоб розробити гнучку архітектуру та використовувати переваги обох підходів. Можна з упевненістю сказати, що зі зростанням рівня складності гнучка концепція всезнаючого «суперінженера» стає дефіцитною, а архітектор програмного забезпечення запрошується компенсувати їхній брак знань.

Результати досліджень чітко свідчать про те, що гнучкі команди відчувають потребу в розширенні гнучких процесів. Про це свідчить набір явних архітектурних практик, типових для традиційної розробки, оцінених експертами як дуже значущих у гнучкій розробці складних програмних рішень. Крім того, можна зробити висновок, що набір архітектурних практик, оцінених як важливі та дуже важливі, узгоджується з практичними архітектурними проблемами, визначеними та класифікованими в роботі. Це підтверджує твердження про те,

що гнучкі команди усвідомлюють, що ці проблеми потребують вирішення, що свідчить про зміну їхнього ставлення до цієї проблеми.

Результати дослідження, проведеного практиками, свідчать про те, що гнучкі команди вважають предметну область важливим фактором у визначенні архітектурної стратегії. Якщо бути точним, то складність предметної області, а також знання членів команди домену потрапляють до факторів, які визначають, скільки попередніх архітектурних рішень потрібно буде прийняти в проекті та скільки часу команді потрібно буде витратити на архітектурний аналіз системи.

Емпіричні результати дослідження показують, що гнучкі команди намагаються знайти власні рішення для модифікації гнучких процесів. Проблема встановлення балансу між гнучкими процесами та традиційними архітектурними практиками у розробці складних програмних рішень вимагає спільних зусиль практиків та дослідників. Незважаючи на те, що протягом останніх років інтерес до цієї теми зріс, все ще можна зробити висновок, що дослідницькі статті, які повторюють цю тему на основі емпіричних висновків, все ще нечисленні.

Практики Agile не тільки визнали необхідність включення явних архітектурних практик у процеси гнучкої розробки, але також вказали на явні архітектурні практики, які вони вважають придатними для включення.

Очевидно, що гнучкі процеси приймають все більше і більше елементів традиційного розвитку, звідси і термін «традиціонізація». Основні цінності гнучкої розробки, встановлені в Agile Manifesto, як і раніше цінуються високо. Однак, з точки зору гнучкої розробки складних систем, риси традиційних методів, такі як структуровані процеси, використання спеціальних інструментів, документація та розробка, керована планом, отримують визнання. Результати, представлені в цій роботі, заохочують до подальших зусиль щодо пошуку рішення для інтеграції явних архітектурних практик у гнучку розробку, але переконавшись, що цінності, які відрізняють гнучкі процеси, залишаються повністю збереженими.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kharchenko, A., Raichev, I., Bodnarchuk, I., & Matsiuk, O. (2021, October). The Survey of Global Software Design Processes. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T) (pp. 291-294). IEEE.

2. Програмна архітектура в розподілених командах гнучких проєктів / О. Гузеляк, Ю. Шевчук, Б. М. Береженко, Ігор Орестович Боднарчук // Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології“, 7–8 грудня 2022 року. – Т. : ТНТУ, 2022. – С. 110–112.

3. Проєктування архітектури програмних систем в проєктах з гнучкими методами управління / І. Боднарчук, О. Харченко, Б. Хоміцький, Г. Шимчук // Матеріали XXI наукової конференції ТНТУ ім. І. Пулюя, 16-17 травня 2019 року. – Т. : ТНТУ, 2019. – С. 46–48.

4. Bodnarchuk, I., Lisovyi, V., Kharchenko, O., & Galai, I. (2018, September). Adaptive Method for Assessment and Selection of Software Architecture in Flexible Techniques of Design. In 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 292-297). IEEE.

5. Imache, Rabah, Said Izza, and Mohamed Ahmed-Nacer. "An enterprise information system agility assessment model." Computer science and information systems 9.1 (2012): 107-133.

6. McDermid, John Alexander. "Complexity: concept, causes and control." Proceedings Sixth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS 2000. IEEE, 2000.

7. Kruchten, Philippe. "Scaling down large projects to meet the agile sweet spot." IBM developer Works 13 (2004).

8. Ambler, Scott, and Mark Lines. "Introduction to disciplined agile delivery." Project Management Institute, 2020.

9. Matković, Predrag, Pere Tumbas, and Marton Sakal. "The RSX model: traditionalisation of agility." Strategic Management 16.2 (2011): 74-83.

10. Babar, Muhammad Ali. "Making software architecture and agile approaches work together: Foundations and approaches." Agile Software Architecture. Morgan Kaufmann, 2014. 1-22.

11. Nord, Robert L., and James E. Tomayko. "Software architecture-centric methods and agile development." IEEE software 23.2 (2006): 47-53.

12. Matkovic, Predrag, et al. "Traditionalisation of agile processes: architectural aspects." Computer Science and Information Systems 15.1 (2018): 79-109.

13. Miles, Matthew B., and A. Michael Huberman. Qualitative data analysis: An expanded sourcebook. sage, 1994.

14. Friedrichsen, Uwe. "Opportunities, threats, and limitations of emergent architecture." Agile Software Architecture. Morgan Kaufmann, 2014. 335-355.

15. Cleland-Huang, Jane, Adam Czauderna, and Mehdi Mirakhorli. "Driving architectural design and preservation from a persona perspective in agile projects." Agile Software Architecture. Morgan Kaufmann, 2014. 83-111.

16. Jeon, Sanghoon, et al. "Quality attribute driven agile development." 2011 Ninth international conference on software engineering research, management and applications. IEEE, 2011.

17. Brown, Nanette, Robert Nord, and Ipek Ozkaya. "Enabling agility through architecture." CrossTalk 23.6 (2010): 12-17.

18. Isotta-Riches, Ben, and Janet Randell. "Architecture as a Key Driver for Agile Success: Experiences At Aviva UK." Agile Software Architecture. Morgan Kaufmann, 2014. 357-374.

19. Weitzel, Balthasar, Dominik Rost, and Mathias Scheffe. "Sustaining agility through architecture: Experiences from a joint research and development laboratory." 2014 IEEE/IFIP Conference on Software Architecture. IEEE, 2014.

20. Waterman, Michael, James Noble, and George Allan. "How much architecture? Reducing the up-front effort." 2012 Agile India. IEEE, 2012.

21. Buchgeher, Georg, and Rainer Weinreich. "Continuous software architecture analysis." Agile Software Architecture. Morgan Kaufmann, 2014. 161-188.

22. Blair, Stuart, Richard Watt, and Tim Cull. "Responsibility-driven architecture." IEEE software 27.2 (2010): 26-32.

23. van der Ven, Jan Salvador, and Jan Bosch. "Architecture Decisions: Who, How, and When?." Agile Software Architecture. Morgan Kaufmann, 2014. 113-136.

24. Kruchten, Philippe. "Software architecture and agile software development: a clash of two cultures?." Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. 2010.

25. Hadar, Ethan, and Gabriel M. Silberman. "Agile architecture methodology: long term strategy interleaved with short term tactics." Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. 2008.

26. Faber, Roland. "Architects as service providers." IEEE software 27.2 (2010): 33-40.

27. Hadar, Irit, and Sofia Sherman. "Agile vs. plan-driven perceptions of software architecture." 2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE). IEEE, 2012.

28. Madison, James. "Agile architecture interactions." IEEE software 27.2 (2010): 41-48.

29. Hopkins, Richard, and Stephen Harcombe. "Agile architecting: enabling the delivery of complex agile systems development projects." Agile software architecture. Morgan Kaufmann, 2014. 291-314.

30. Nord, Robert L., and James E. Tomayko. "Software architecture-centric methods and agile development." IEEE software 23.2 (2006): 47-53.

31. Boehm, Barry, et al. "Architected agile solutions for software-reliant systems." Agile Software Development: Current Research and Future Directions (2010): 165-184.

32. Nord, Robert L., Ipek Ozkaya, and Raghvinder S. Sangwan. "Making architecture visible to improve flow management in lean software development." IEEE software 29.5 (2012): 33-39.

33. Hinsman, Carl, Neeraj Sangal, and Judith Stafford. "Achieving agility through architecture visibility." Architectures for Adaptive Software Systems: 5th International Conference on the Quality of Software Architectures, QoSA 2009, East Stroudsburg, PA, USA, June 24-26, 2009 Proceedings 5. Springer Berlin Heidelberg, 2009.

34. Keuler, Thorsten, Stefan Wagner, and Bernhard Winkler. "Architecture-aware programming in agile environments." 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture. IEEE, 2012.

35. Tyree, Jeff, and Art Akerman. "Architecture decisions: Demystifying architecture." IEEE software 22.2 (2005): 19-27.

36. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин МОЗ України від 10.12.1998 № 7. // Офіційний сайт Верховної Ради України. – [Електронний ресурс]. – Режим доступу https://zakon.rada.gov.ua/rada/show/v0007282-98

37. Бідяк О. Профілактика отруєння хлором. // Офіційний сайт управління держпраці в Тернопільській області. – [Електронний ресурс]. – Режим доступу https://te.dsp.gov.ua/profilaktyka-otruyennya-hlorom/

ДОДАТКИ

УДК 004.41

А. Вивюрка, Л. Мариненко, О. Нога, Б. Хоміцький, Т. Ланевич
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСІВ CI/CD В ГНУЧКИХ ТЕХНОЛОГІЯХ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

A. Vyviurka, L. Marynenko, B. Khomitskyi, O. Noha, T Lanevych

RESEARCH OF THE EFFICIENCY OF CI/CD PROCESSES IN AGILE SOFTWARE DEVELOPMENT TECHNOLOGIES

Процеси безперервної інтеграції (Continuous Integration – CI) та безперервного розгортання (Continuous Deployment – CD) були популяризовані наприкінці 90-х як частина eXtreme Programming [1], що загалом належить до гнучких (Agile) технологій розробки [2]. Поточні IT-стратегії значною мірою залежать від здатності компаній впроваджувати зміни або виправлення гнучким чином та безпомилково. Потреба в автоматизації різко зростає з розвитком технологій, оскільки останні десятиліття методи водоспадної розробки програмного забезпечення були замінені гнучкими підходами до розробки програмного забезпечення.

В даній доповіді пропонується фреймворк для дослідження проєктів з розробки програмних продуктів з метою оцінювання ефективності процесів розробки з впровадженням CI/CD. Для досягнення цієї мети пропонується оцінювати процеси за такими характеристиками.

1. Можливість впровадження технологій Agile в процеси тестування.
2. Ефективність комунікації в команді та в рамках проєкту між командами.
3. Збільшення продуктивності роботи розробників.
4. Збільшення передбачуваності проєкту.

Можливість впровадження гнучких технологій тестування завдяки гнучкій інтеграції розглядалась в роботі [3]. Безперервна інтеграція вважається важливою для підтримки гнучкого тестування. Вважається, що гнучке тестування включає в себе такі практики, як визначені клієнтом приймальні (acceptance) тести, автоматизація цих тестів і їх виконання в наборі регресійних тестів щонайменше щодня, а також розробка модульних (unit) тестів для всього нового коду під час кожної ітерації (спринту) з наступним виконанням цих модульних тестів з кожною збіркою. Задача дослідження полягатиме виявленні того, чи використовуються на проєкті модульні тести з їх автоматичним виконанням.

Задача оцінки ефективності спілкування полягає в тому, щоби при дослідженні проєкту інформація про дефекти автоматично формувалась на надсилалась усім зацікавленим сторонам включно з розробником. Тобто даний пункт досліджень логічно побудований на наявності автоматичних модульних тестів та необхідності застосування процесів інтеграції. Тоді при невдалому завершенні будь-якого тесту формуватиметься відповідний звіт та надсилатиметься тому, хто створив частину програмного коду, котра спричинила збій. Тобто проєктний менеджер чи тестувальник не затрачатиме час на формування таких звітів і їх розсилку. Особливо це актуально, коли над проєктом працює декілька команд, і коли часка затрат комунікацію та узгодження процесів розробки між командами значно зростає.

Безперервна інтеграція сприяє збільшенню продуктивності розробників завдяки очевидній можливості паралельної розробки, тобто роботи декількох програмістів над одним і тим же програмним кодом. Як наслідок, зменшується час на компіляцію та тестування кожним розробником та надає йому можливість впровадити більше нових властивостей (вимог) та змін. Крім того, в роботі [4] розраховано чистий прибуток від

впровадження безперервної інтеграції шляхом вимірювання часу, зекономленого завдяки тому, що розробники не здійснювали компіляцію вручну та не проводили тестування перед кожною операцією коміту, оскільки ці процеси виконувались автоматично в рамках безперервної інтеграції. Це означає, що основна перевага постійної інтеграції полягає в економії часу для розробників.

Можливість точнішої оцінки прогресу проєкту та термінів його завершення відносно проміжних етапів та кінцевого терміну відповідно розглядалась, наприклад, в роботах [5] та [6]. Актуальна інформація про покриття продукту тестами, відсоток успішно виконаних та кількість дефектів дозволяє оперативно вживати управлінські заходи, як то зміна пріоритетів, перерозподіл завдань, залучення необхідних ресурсів, та дотримуватись графіку виконання проєкту. Зокрема, проєктний менеджер матиме змогу виявляти проблеми на ранніх етапах, запроваджувати раннє тестування нефункціональних вимог.

Таким чином, впровадивши даний фреймворк, можна досягнути позитивних результатів завдяки впровадженню процесів CI/CD у Agile-проєктах. Ми дійшли висновку, що існує не одна, а декілька переваг постійної інтеграції. Для кожного проєкту можна виявити взаємозв'язок між безперервною інтеграцією та гнучкими методами тестування автоматизованих тестів клієнта та написанням модульних тестів у поєднанні з новим початковим кодом. При цьому недослідженими залишаються наступні питання. Наприклад, чи підтримує безперервна інтеграція практику модульного тестування, чи, навпаки, вони підтримують один одного. Також важко відокремити вплив безперервної інтеграції на практику автоматизованого α- та β-тестування від контекстних факторів (таких як організаційна структура, культура та доступність клієнтів).

Питання підвищення продуктивності розглянуто в аспекті збільшення кількості завдань, виконаних розробником. Проте його продуктивність в контексті обсягу написаного програмного коду залишається незмінною. Тобто економія часу не зовсім очевидна в кожному проєкті. Звичайно, ефекти, які обговорюються в цій статті, не становлять вичерпний перелік переваг постійної інтеграції. Однак можна стверджувати, що краще розуміння потенційних відмінностей у реалізаціях CI/CD-процесів та їхніх наслідків може допомогти проєктам сформувати свою безперервну інтеграцію таким чином, щоб оптимізувати переваги, яких вони прагнуть досягнути.

Література

1. Beck, K. (2000). Extreme Programming Explained Addison-Wesley. Reading.

2. Agile alliance: manifesto for agile software development. Available at: http:// agilemanifesto.org [retrieved 10.10.2022].

3. Stolberg, S. (2009, August). Enabling agile testing through continuous integration. In 2009 agile conference (pp. 369-374). IEEE.

4. Miller, A. (2008, August). A hundred days of continuous integration. In Agile 2008 conference (pp. 289-293). IEEE.

5. Goodman, D., & Elbaz, M. (2008, August). "It's Not the Pants, it's the People in the Pants" Learnings from the Gap Agile Transformation What Worked, How We Did it, and What Still Puzzles Us. In Agile 2008 conference (pp. 112-115). IEEE.

6. Liu, H., Li, Z., Zhu, J., Tan, H., & Huang, H. (2009, July). A unified test framework for continuous integration testing of SOA solutions. In 2009 IEEE International Conference on Web Services (pp. 880-887). IEEE.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.41

Володимир Чичук, Леонід Мариненко, Володимир Сенківський, Богдан Хоміцький, Тимофій Ланевич
Тернопільський національний технічний університет імені Івана Пулюя

ОЦІНЮВАННЯ АЛЬТЕРНАТИВНИХ ПРОГРАМНИХ АРХІТЕКТУР МЕТОДОМ АНАЛІЗУ ІЄРАРХІЙ

Volodymyr Chychuk, Leonid Marynenko, Volodymyr Senkivskyi, Bohdan Khomitskyi, Oleksandr Noha, Tymofii Lanevych

EVALUATION OF ALTERNATIVE SOFTWARE ARCHITECTURES WITH ANALYTIC HIERARCHY PROCESS

З ускладненням програмних систем стає все важче оперативно дотримуватись відповідності вимогам якості на етапі їх створення. Для зменшення цього негативного впливу цей процес переносять на початкові стадії проектування, особливо під час формування архітектури. Архітектура у цьому контексті означає сукупність елементів, які містять в собі основні обчислювальні завдання та забезпечують їх взаємодію, створюючи конфігурацію.

Процес проектування архітектури включає кілька етапів [5]:

1. Визначення вимог до програмного забезпечення, як функціональних, так і якісних, здійснюється на основі аналізу потреб усіх зацікавлених сторін.

2. Вибір альтернативних проектних рішень. На основі вимог створюються різні варіанти проектування, які потім порівнюються для знаходження найбільш відповідного.

3. Аналіз і оцінка проектних рішень. Кожен варіант проектного рішення повинен бути оцінений та порівняний з іншими, з урахуванням їх впливу на виконання якісних аспектів.

4. Загальний аналіз архітектури та прийняття рішення. З урахуванням попередніх етапів архітектор обирає оптимальний варіант, який задовольняє всі вимоги якості. Якщо такого варіанту немає, то досліджуються конфлікти між критеріями якості, шукаються компроміси для вибору оптимального рішення. Розглянемо питання оцінювання якості архітектури по множині показників якості методом аналізу ієрархій (MAI) з використанням оптимізаційного алгоритму визначення ваг альтернатив.

Процес порівнювального оцінювання архітектур з використанням MAI представлено на рис.1.

Вибір архітектури повинен виконуватись таким чином, щоб побудована на її основі програмна система (ПС) задовольняла вимогам якості. Тому тут представлено два рівні взаємопов'язаних критеріїв якості:

– критерії якості ПС у відповідності зі стандартом ISO/IEC 25010;
– критерії якості архітектури;
– альтернативні архітектурні рішення.

Множина критеріїв якості ПС $\{K_i^1\}$, та обмеження на них визначаються при розробці вимог до ПС. А множина критеріїв якості архітектури $\{K_i^2\}$ визначається з використанням методу QFD (Quality Function Deployment, або методу парних порівнянь [1].

Необхідно вибрати з наявних альтернатив таку, яка б найкраще забезпечувала якість ПС, тобто треба вирішити задачу оптимізації за сукупністю критеріїв $\{K_i^1\}$, $\{K_i^2\}$. Це задача багатокритеріальної ієрархічної оптимізації і для її розв'язання будемо використовувати метод аналізу ієрархій Сааті [2].

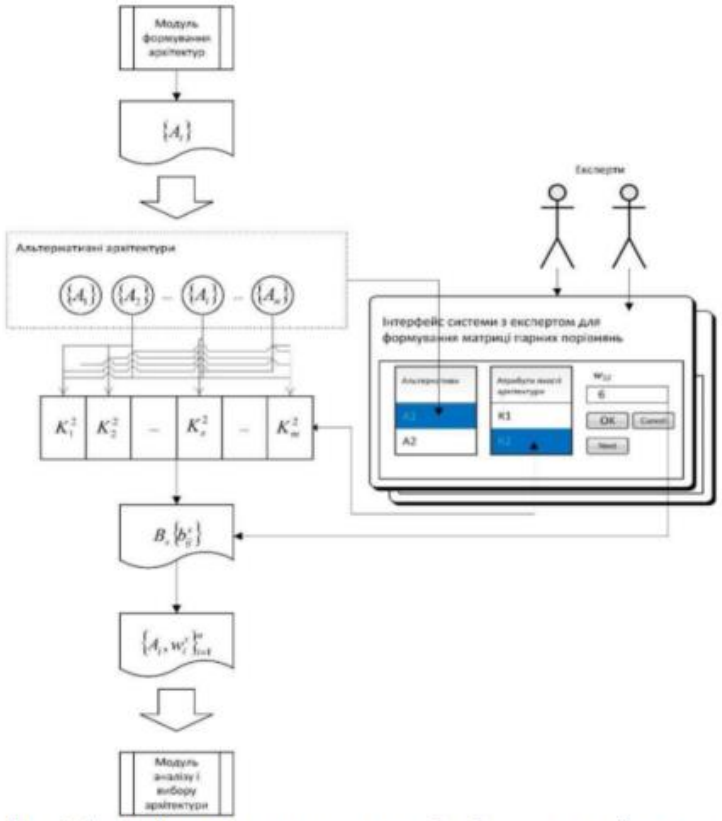

Рис. 1. Ієрархічне представлення задачі оцінювання архітектури

При використанні MAI для рішення таких задач відносні оцінки критеріїв (ваги) $\{\omega_i^S\}$ для альтернатив на кожному рівні знаходяться з використанням матриць парних порівнянь $B^S(b_{ij}^S)$, які заповнюють експерти (тут b_{ij} звизначає перевагу i-тої альтернативи над j-ю по реалізації s-го критерію).

Коефіцієнти матриць повинні бути узгодженими, тобто $b_{ij} = w_i / w_j \quad \forall b_{ij} \in B$. Ваги в цьому випадку знаходяться як компоненти власного вектору матриці парних порівнянь, які відповідають максимальному характеристичному числу матриці. Обчислення власного вектору матриці є досить трудомісткою процедурою. Тому користуються як правило наближеними співвідношеннями [2]

$$w_i = \frac{1}{n}\sum_{j=1}^{n}\frac{b_{ij}}{\sum_{j=1}^{n} b_{ij}} \qquad (1)$$

Але при значній кількості альтернатив, в силу дії на експертів різних негативних факторів, матриця є неузгодженою і її ранг буде відмінним від одиниці, тобто матриця буде мати декілька власних значень. А.Павловим в роботі [3] для розв'язку даної задачі запропоновано моделі для різних форм представлення міри неузгодженостей і метод обчислення ваг альтернатив з умови мінімізації неузгодженості матриці. А в роботах [4],

[5] цей метод було використано в задачі вибору архітектури ПС з врахуванням показників якості. В даній задачі було використано міру неузгодженості наступного виду.

$$\left|\frac{w_i}{w_j} - b_{ij}\right| \le \delta_{доп} \cdot b_{ij}, \quad \delta_{доп} \ge 0, \tag{2}$$

де $\delta_{доп}$ – задане порогове значення.

Тоді ваги, які мінімізують (2), знаходяться з рішення задачі лінійного програмування:

$$\begin{aligned} &\min_{\{w_i\}} \sum_{i=1}^{n}\sum_{j=1}^{n}\left(y_{ij}^{+} - y_{ij}^{-}\right) \\ &w_i \ge a_i, \quad i = \overline{1,n}, \\ &w_i - b_{ij} w_j = y_{ij}^{+} - y_{ij}^{-}; \qquad y_{ij} \ge 1; \end{aligned} \tag{3}$$

$$\begin{aligned} &-\delta_{доп} \cdot b_{ij} \cdot w_j \le w_i - b_{ij} \cdot w_j \le \delta_{доп} \cdot b_{ij} \cdot w_j, \\ &y_{ij}^{+}, y_{ij}^{-} \ge 0; \quad i, j = \overline{1,n}. \end{aligned} \tag{4}$$

Таким чином, знаходження ваг альтернативних архітектур зводиться до рішення задачі (3), (4). При розв'язуванні даної задачі може виявитися несумісність обмежень (4), в такому випадку потрібно збільшити $\delta_{доп}$. Використання наведеного алгоритму дає змогу отримати ваги альтернатив як по реалізації кожного критерію якості, так і по їх сукупності. Однак, при прийнятті рішення з вибору архітектури по отриманих оцінках ω_i^S для врахування обмежень та пріоритетів розробників потрібно аналізувати конфлікти між критеріями якості та шукати компромісні рішення.

Література

1. Харченко О.Г. Інструментальний засіб розробки та комунікації вимог якості до програмних систем [Текст] / О.Г. Харченко, В.В. Яцишин, І.Е. Райчев // Інженерія програмного забезпечення. – 2010. – № 2. – с. 29–34.

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

3. Павлов А.А. Математические модели оптимизации для нахождения весов объектов в методе парных сравнений. Павлов А.А, Лищук Е.И., Кут В.И. // Системні дослідження та інформаційні технології – К.: ІПСА,– 2007. №2, с. 13 –21.

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

5. Harchenko, Alexandr, Ihor Bodnarchuk, and Iryna Halay. "Decision support system of software architect." 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS). Vol. 1. IEEE, 2013.